University of Portsmouth

BSc (Honours) Degree in Computer Science

Final Year Project

Improving the Ease of Use on Smart Watches, with Gesture Control

By

Max Edmund Hunt

Project unit: PJE40
Supervisor: Dr Jacek Kopecky

April 2015

Computing and Mathematics Programme Area

Abstract

Wearable devices are ever more present in today's technology market. These devices integrate with smart phones with the intent to make our data more accessible. However, the way we interact with that data is still the same on wearable devices, tapping on a screen or pressing buttons. In order for wearable technology to improve the ease of use of our smart devices, a technology must be found that is natural and easy for a user to operate. This project suggests gesture control as the solution. By building a library, Gebble.js, that implements gesture control for the Pebble smart watch, it allows developers to become more involved with gesture control. Instead of creating just another application, they could build a better user experience utilising a more natural way for consumers to interact with their data.

Acknowledgements

I would like to thank Dr Jacek Kopecky for his constant support and patience throughout this project.

A special mention goes to David Gauld and Mike Williams for their constructive criticism and my father, Ian Hunt, for his continued support.

Contents

1 Introduction

For years, the technology industry has been working towards a new easy-to-use device that communicates with the connected world. Wearable technology is now testing the water as an answer to this innovative direction. With the recent announcements of the Pebble Time and Apple Watch and the currently available Pebble Watch and Moto 360, the smart watch is becoming more popular (Figure 1).

However, are they actually improving the ease of use and minimising the un-natural interaction between a user and their phone? This project is exploring the fact that this assumption may not be the case. By swapping the tapping on said phone to tapping on a smart watch, in the worst case scenario, another step is being created when checking your messages and consuming the battery of a user's device(s). How do you then introduce a more natural way to interact with a phone?

Using the Pebble smart watch as a platform for my project, this project is implementing a Gesture Control library, as a proof of concept as well as opening the doorway to developers and a new range of gesture based applications. This reduces the need for endless tapping and button clicking as the ease of use is improved by predetermined gestures. Working in JavaScript, enabled in the 2.0 update by the Pebble team, and employing a method of gesture detection, I hope to prove that the ease of use in smart watches, can be greatly improved with the use of gesture control.

The Pebble presents unique challenges, such as memory availability and processing power, that this project overcame, but it also improves the chance of this project being used in a wider context. At 700,000 sales, see figure 1, the library has implications across a wide number of professions, from medical to recreational. For example, the applications range from, a way to interact with your phone for the visually impaired to a dance move detection application. Later in the chapter "Case Study Design and Development," I explain the choice of the Pebble platform in more depth.

Figure 1 - Kunze, E. (2015). *Report: Top 10 Smartwatch Manufacturers By Sales (Q4 2014).*

1.1 Aims and Objectives, Constraints

This section will outline my aims for this project and the objectives I will have to achieve to meet this aim. It also formally describes what the planned deliverables are but moreover, what may hold back development and the meeting of the aims and objectives in terms of technical and physical constraints.

1.1.1 Project Aim and Objectives

Considering the statement of the problem and my proposed solution of the problem, the overall aim of the project is to improve the ease of use on smart watch platforms, using JavaScript as the developing language. This will be achieved in the form of a gesture control library, available to the developing community around the pebble smart watch.

In order to achieve this I will be completing these objectives:

- Investigate **Gesture Control** to find un-complicated gestures to apply initially in the applications.
- Research and scrutinize different **Gesture Recognition** techniques that have already been implemented and identify if any current development is adaptable or transferable to JavaScript.
- Explore and apply **Gesture Control** to the Pebble using the 3D accelerometer.
- Consider the creation of a website to create a **Distributed System** across the platform to store data and/or application features.
- Meet the Pebble's memory requirement by generating/developing **Efficient Code**.
- Make sure all data collected is in accordance with the **Data Protection Act**.
- Release the applications and or library to the developer community and Pebble App Store

1.1.2 Project Deliverables

These are to formally stated deliverables I hope to create to achieve my overall aim:

- A Library for Gesture Control, written in JavaScript
- A number of native applications for the Pebble.

Additionally, depending on the outcome of initial research and planning, I will produce the system artefact:

- A website interfacing with the applications; used to store data and for additional computing power if needed

Finally, documentation will be produced. This documentation will outline how to use the library and any applications that are created to achieve the aim of the project. It may come in the form of website entries, commenting in the developed code or and a standalone document.

1.1.3 Project Constraints

With every project there are constraints and this project is no exception. Undoubtedly, the generic constraints of time; the project has only 8 months, and workload; I personally have other units and coursework that I need to complete, that will affect the outcome. However, other constraints are in play that remain unique and somewhat uncommon in the now very much evolved, industry of technology; the most pronounced being memory availability, but also industry evolution and the Pebble SDK platform.

The introduction already briefly mentioned the fact that the Pebble, being a small gadget in nature, has very limited space for memory storage. As a consequence, it not only has a much smaller available space for volatile memory but also non-volatile. The pebble can only have 8 apps stored on it at a time, each with a maximum memory allowance. A solution has already been thought of and is incorporated into my objectives and deliverables. This solution is the storing of data on an external website, which can then interact with the Pebble in the form of Ajax requests.

Additionally, with the industry of technology and its constant evolution, the wearable device sector is growing very quickly, Figure 2. This could potentially mean that the platform I have opted for could be a legacy device by the time of this projects completion and, in fact, the Pebble has announced an upgrade to my chosen platform, this is the Pebble Time for the near future.

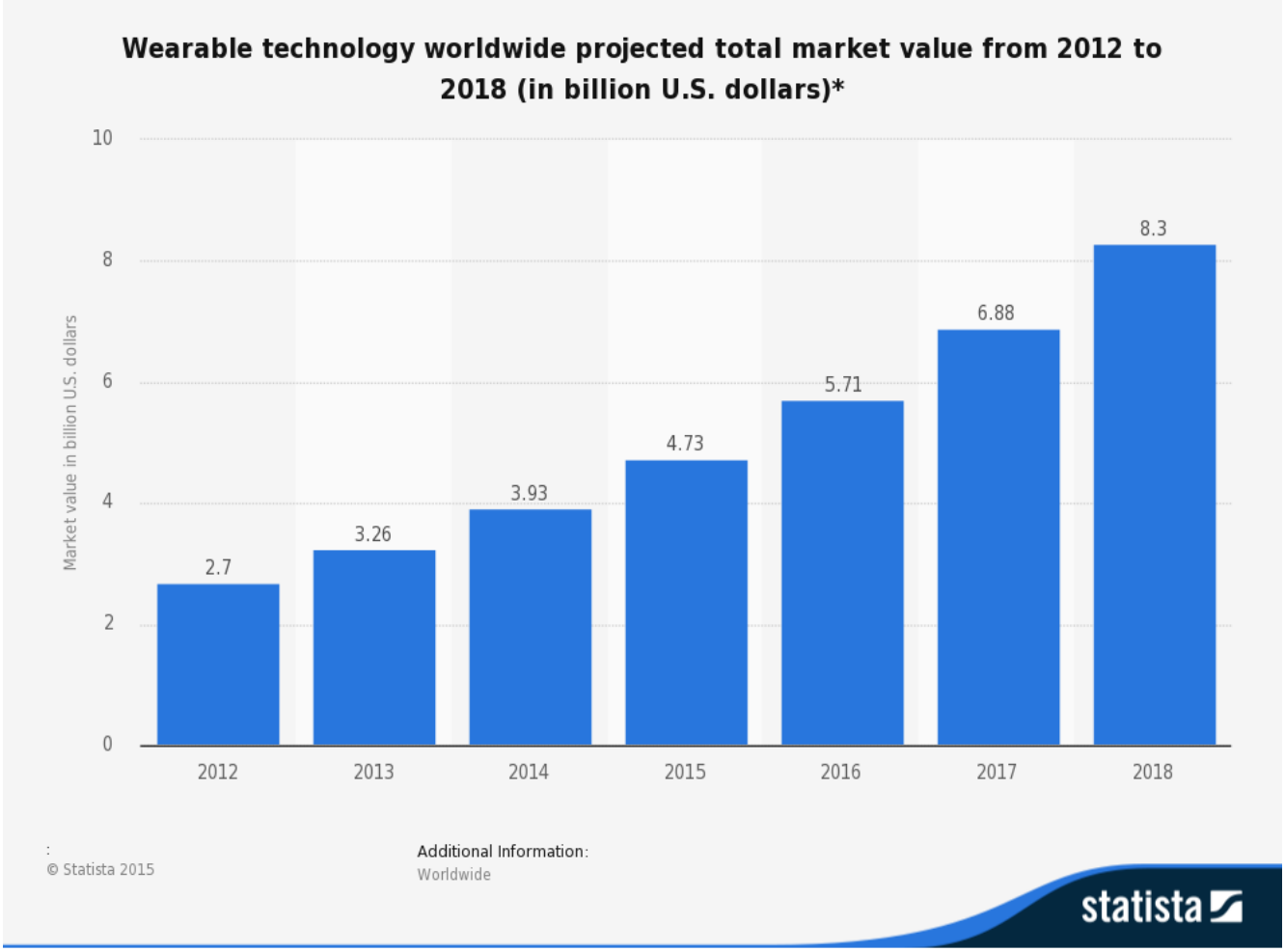


Figure 2 - Statista. (2015). Wearable technology worldwide projected total market value from 2012 to 2018 (in billion U.S. dollars)

2 Literature Review

The overall aim of this literature review to gain and evaluate current knowledge in the area of wearable technology, within the field of gesture control. The review will look at literature in different topics, namely gesture recognition, usability/ease of use, current applications and the privacy and security implications that could have effect on this project.

2.1 Gesture Recognition and General Motivation

“Presently, wearable technology is receiving significant attention worldwide,” (Park, Hwang & Moon, 2014, p. 33) and also provides new possibilities for interacting with various applications. (Kela, Korpipää, Mäntyjärvi, Kallio, Savino, Jozzo & Marca, 2005, p. 285)

Gesture control can be considered a natural communication channel, which has not yet been fully utilised in human–computer interaction. (Kela et al, 2005, p.285) However, the choice of the algorithms to implement this depends on the environment in which the system is required to operate. (Hackenberg, McCall & Broll, 2011, p.20)

Wang, Zhang and Dai (2007, p. 238), use image manipulation and particle clustering to find gestures within images and present data that proves this method to be best. However, Hackenberg et al (2011, pp. 20-26) state and prove their method of gesture recognition is best using distance and colour filters. There is no one correct way to go about gesture recognition. The pebble operates over the 3D accelerometer, therefore image processing is not possible but there are techniques using mathematical transposition to calculate vectors(Wu, Pan, Zhang, Qi & Li, 2009, p.25-38) .

The pebble is powered by an ARM Cortex-M3 processor, an even though it is the industry-leading 32-bit processor (“Overview for ARM Cortex-M3 Core”, n.d.), is also designed to be cost efficient. Calculating vectors through mathematical solutions maybe beyond it and therefore a solution incorporating current techniques from Wu et al (2009, p.25-38) should be implemented.

2.2 Ease of Use and Perceived Ease of Use

Improving the ease of use is the target of this project, or is it?. Yang, Lee, Park and Lee (2014, p.166) and Davis (1989, p. 320), both accept a difference between the perceived ease of use and the actual ease of use. Davis proposed a model in which to gauge this difference, in technology, adequately named the “Technology Acceptance Model.” However, since then, Legris, Ingham and Collerette (2003) have criticised this model. This is covered in the next section. The relevance this model has to gesture control, is the fact that although many more applications are being integrated with gesture control (Tran &

Oh, 2014, p. 298), is it actually making their product or service easier to use and is using the Technology Acceptance Model the best way to access this.

2.2.1 Technology Acceptance Model and Using It

There are 2 main variables, which are the building blocks to the Technology Acceptance Model.

Davis says that (1989, p. 320), “people tend to use or not use an application to the extent they believe it will help them perform their job better.” This is the first variable, perceived usefulness. The second is the perceived ease of use, defined by when “potential users believe that a given application is useful, they may, at the same time believe that the system(s) is too hard to use and that the performance benefits of usage are outweighed by the effort of using the application.”

Davis later then revised this into the Technology Acceptance Model 2 (Legris et al, 2003). “It (now) includes subjective norms, and was tested with longitudinal research designs.” However, Legris (2003) found this model to be successful at predicting the perceived usefulness and the perceived ease of use in 40 percent of a system and the results, over multiple systems, were neither clear nor consistent. This data also reflected the findings of Hu, Chau, Liu Sheng, and Tam (1999, p. 104) at an average of 44 percent.

Hu et al, (1999, p. 104) came to this result via a comparison of the perceived usefulness and perceived ease of use using the study and then via a number of physicians in there study.

Legris et al(2003) thought it was due to the multiple errors in research methods and practices. These errors are that many of the studies involved students and were not in a business environment, most were on the introduction of office systems not a range of business applications and all were self-reported rather than completed by a third party.

The Technology Acceptance Model and the Technology Acceptance Model 2, in deduction, is useful but a better method of the detection of perceived usefulness and perceived ease of use should be completed. This is yet to be fully identified.

2.2.2 A non-theoretical gesture control example

However, Silva and Rodrigues (2014, p 287) state that they initiated “usability testing with users to determine performance measures, the influence of gesture control on perceived usefulness and ease of use of the system.”

It found that in their medical application of gesture control “participants are able to perform the tasks of search, selection and manipulation of 2Dimages and 3D models, quickly and accurately, demonstrating the usefulness of the system as a possible effective and competitive alternative, [to previous approaches] for gesture based interactive control of medical images.”

In conclusion, although the Technology Acceptance Model may fall short of what is required of it, perceived ease of use and or usefulness can be proved, or not be proved, within a system of technology through a real-life study rather than a theoretical one.

Furthermore, gesture control has been proved, with or without this model, to be an effective evolution in the ease of use field.

2.3 Privacy and Security

Due to the release of Google Glass, the public is now highly aware of wearable technology (Kirkham and Greenhalgh, 2015, p. 26), along with its privacy and security concerns. With adoption “expected to surge when the price of Google’s Glass device drops to as low as £150 by 2018” O'Flaherty (2014, p23) and given the constant advances in technology, the cyber-security and privacy sector needs continuous innovation, legislation and investment to keep pace. (Tyer, 2015).

The Data Protection Act 1998, part 2 section 7, covers the personal data use of consumers by business as well as the collection of this data and how a consumer has right to obtain that data in its raw form. But due to the nature of wearable devices, this data is processed and transferred to other devices before this can happen. “This lack of clarity is something that manufacturers may wish to address by ensuring that they are capable of providing access to unprocessed data (if requested),” McDonald (2015, para 6).

The European parliament’s board that improves this legislation, the Science and Technology Assessment Panel (STOA), are discussing this issue with legislators. “The aim of STOA projects is to provide parliamentarians with an overview of important emerging science and technology developments that are not yet in the focus of European legislation,” (Böhle, Coenen, Decker, Rader, 2012, p71). Regulation for this type of privacy has not been introduced yet but has been drafted for use in 2015, (McDonald, 2015, para 6) in the form of the recently released, by the European Commissions, draft of the General Data Protection Regulation. (De Hert & Papakonstantinou, 2012, p.130)

However, from the point of view of the consumer, the DPA means “Consumers using wearable technology are unlikely to be in breach as it includes an exemption for the collection of personal information for domestic purposes.” O'Flaherty (2014, p24). This implies, currently, personal uses of devices are legislated correctly. Nevertheless, “one minute on the way into work, it can be personal and the next minute, it’s not” O'Flaherty (2014, p24).

All this means that the use of the library and or applications that use the library, should have the capability to print out the data to the console, should a consumer wish to see the data. Although the Pebble may have less publicity than Google Glass, future legislation may change the way that this project process a user’s information, as gestures are personal data to a user.

3 Methodology

This section is focused on the methodology choice. It explains why my choice of approach was heavily attentive to the prototyping method as well as why the merits of the other choice did not sway me into using them.

3.1 The choice

The chosen methodology has implications across the whole project, from design to testing and is therefore not a decision to be quickly taken. Yet, whilst searching for a methodology that fitted this project type and my development style, I came to the realisation about the choice I was making. This realisation was that these are theoretical methodologies and that one would not be a perfect fit for a real-world project. However, I narrowed the choice to two methods, but decided on prototyping as my main method.

3.1.1 The Agile Manifesto

Nonetheless, the first method I looked into was the predominant approach in Software Engineering (Losada, Urretavizcaya & Fernández-Castro, 2013, p. 2268). This is the Agile Manifesto. A diagram that explains this method is available in figure 3. However, Losada et al (2013, p.2268) also found that this method did not lend itself to design and archetypical issues, which I foresee as a critical part of the development of the library. For this reason alone, the applicability of this method was already questionable, but what tipped the equilibrium is one of the main advantages of this method, the ability to change requirements during the development process.

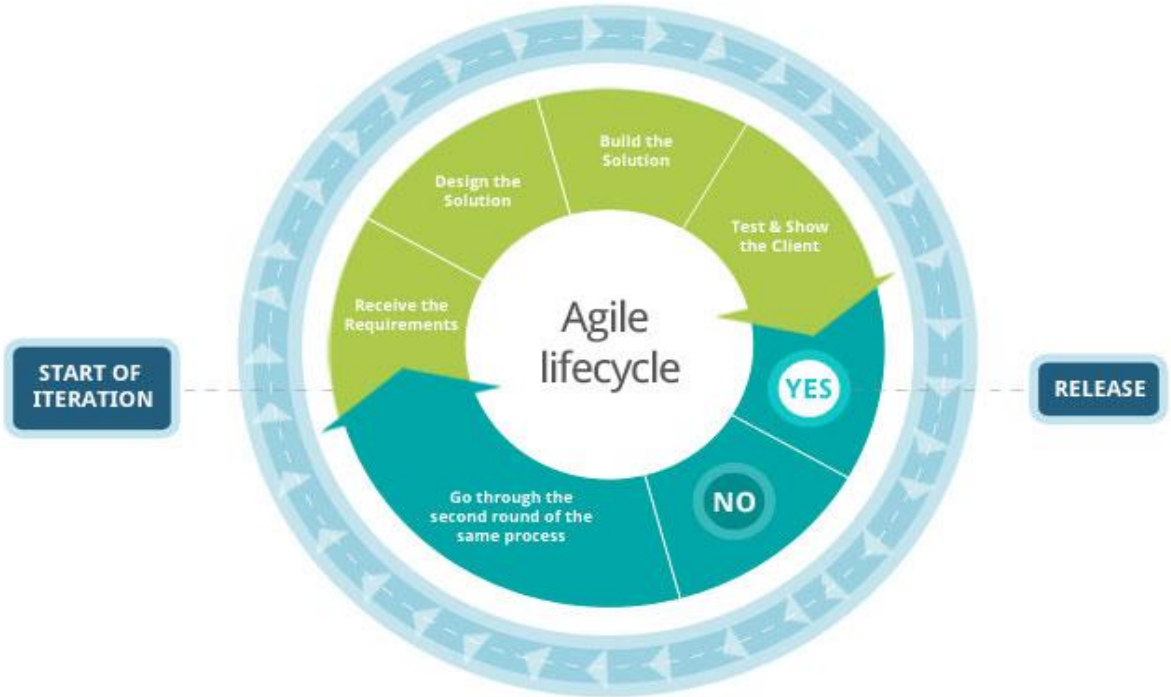


Figure 3 - ("Methodologies", n.d.)

Whilst I do foresee some movement of the requirements, the aim of the project is clear and the requirements of the solution shouldn't change because, for example, the task was harder than expected. If a requirement is not met, an explanation of why it hasn't been met should be reported rather than changing it and then a new requirement stated in the next cycle. This could lead to misdirection from the projects initial aim.

3.1.2 Prototyping

Whilst never having developed in this manner before and with limited literature on the subject, as stated by Öztürk (2013, p.797), the decision was made to use this method for the software development lifecycle. Prototyping allows a gradual increase of complexity over time, whilst not deterring you from your original requirements. It is also stated that pattern recognition projects, such as this one, fit the prototyping technique well. "Prototyping techniques play an important role in many pattern recognition and computer vision tasks," Cordella, Stefano & Fontanella (2007, p.157).

The specific method of prototyping in use will be the evolutionary type. Cordella, Stefano & Fontanella (2007, p.159) explained this technique with an analogy to natural selection. The fittest and most resourceful individuals survive and then pass on their genetic traits and experiences to their offspring. In other words, each prototype has its best features taken and built on into the next prototype and difficulties that became pronounced in the previous stage of development should not happen again. The foremost reason for choosing this prototyping practice was that it allows the comparison of requirement validations to the original expectations and not to the current system functionality (Zhang, Lv, Xu & Mu, 2010, p. 1557).

Another quality that evolutionary prototyping is designed for is easy growth and frequent improvements, (Du Bois & Gerritsen, 2013, p. 138). As well as suiting the project type, this feature of evolutionary prototyping suits the strict timescale, 8 months. On the other hand, this is also one of the drawbacks. Multiple prototypes must be completed in order to justify the use of this method. Upon reaching the end of the timescale with the just one prototype, the end artefact is lot less developed. Conversely, too many prototypes could also be created. Moving on after a small improvement, prototypes are created but the gain of features is reduced resulting in the same conclusion, lack of overall development. Yet, I do not believe that this will happen with the complexity of the project and the natural process of development.

The other main subset of prototyping (Öztürk, 2013, p. 798) is "throw-away" prototyping. Also referred to as abstract prototyping, it operates as you would expect, throwing each prototype away at each stage and starting again from the beginning, using what you have learnt from the previous effort. However, within the literature that was found, it was stated that although it allows for ease of following of functional requirements, it lends itself more to user interface design improvements (Du Bois & Gerritsen, 2013, p. 137). The main aim of this project is to produce a library, a piece of development work which does not have a graphical user interface. Consequently, the main advantage of this technique would have little effect on my development.

3.2 Overall Justification of the choice of the methodology

It was therefore deemed that “throw-away” programming could work, but the evolutionary subset should be used, as it is a better fit. The drawback stated for evolutionary prototyping is eclipsed by the lack of an overall advantage from the throw-away prototyping, but overall, prototyping seems a lot more effective, in terms of a software development life cycle, when this projects aims are considered. The agile method is not a poor method of development, far from it; the advantages are just not applicable to this artefact, and my own, development criteria.

3.3 Project Management - Plan

Using the evolutionary prototype method and taking influence from the gantt chart already produced in the PID (Appendix A). The project development gantt chart can be seen below (figure 4). Due to the methodology, the design, testing and implementation sections overlap on each prototype. This allows the evolution of the prototypes. Chapter 7, the conclusion, will report on the outcome of this project plan.

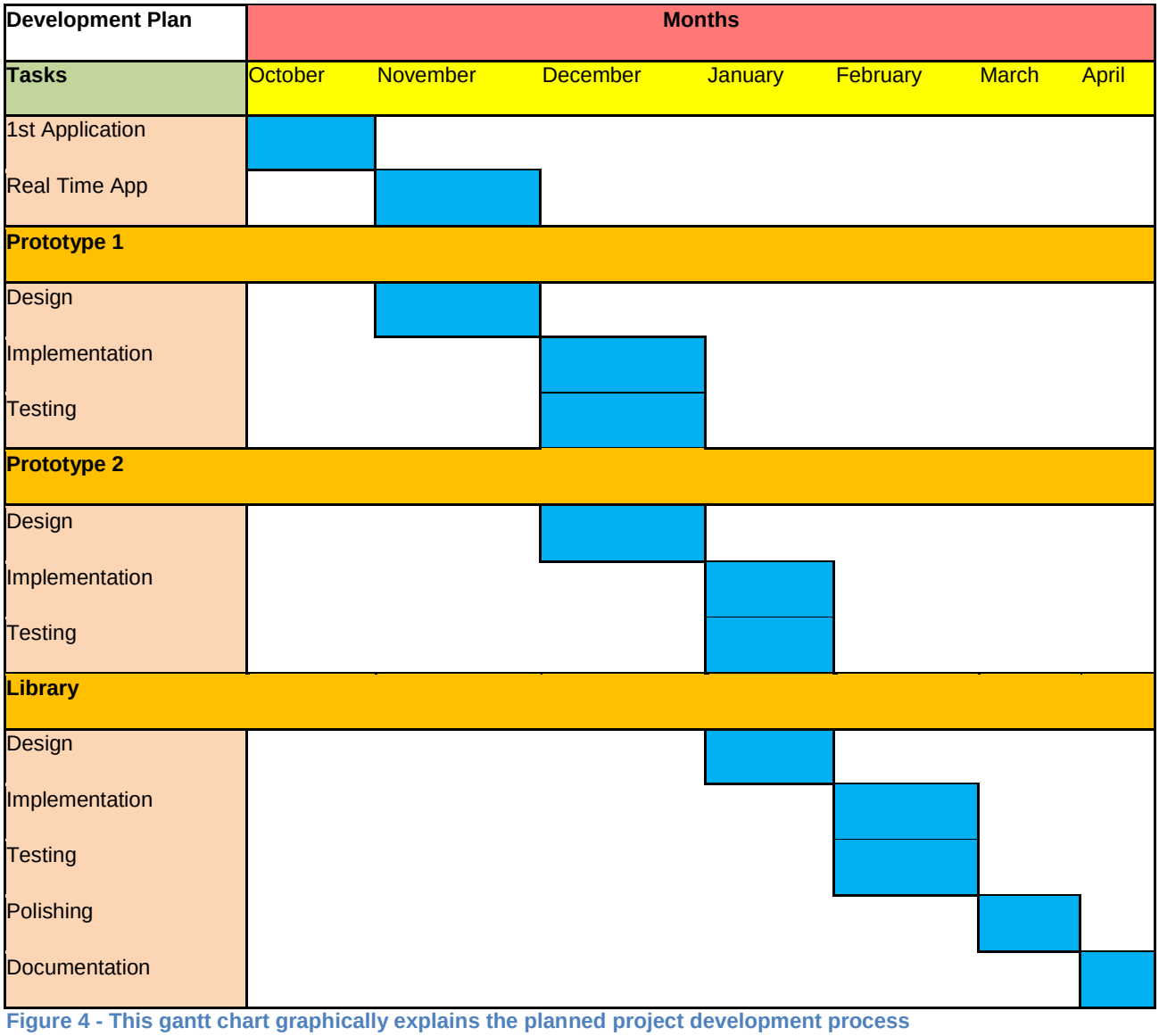


Figure 4 - This gantt chart graphically explains the planned project development process

4 Requirements

This section defines the functional and non-functional requirements for my main artefact, the library, and the prototypes themselves.

Whilst it is necessary to always try your best in order to succeed at the highest level, realistic requirements must be made in order for the project to have an achievable and quantifiable outcome.

These requirements have been made with the aims and objectives (chapter 1.1) in mind as well as the constraints of time frame, technology and feasibility. This information about the constraints can be seen in chapter 1.3 - Project constraints.

Below is the MoSCoW scale. This is the scale, by which I have classed my requirements, (Brennan, 2009).

Letter	Meaning	Description
M	MUST	Describes a requirement that must be satisfied in the final solution for the solution to be considered a success.
S	SHOULD	Represents a high-priority item that should be included in the solution if it is possible. This is often a critical requirement but one which can be satisfied in other ways if strictly necessary.
C	COULD	Describes a requirement which is considered desirable but not necessary. This will be included if time and resources permit.
W	WON'T	Represents a requirement that stakeholders have agreed will not be implemented in a given release, but may be considered for the future. (note: occasionally the word "Would" is substituted for "Won't" to give a clearer understanding of this choice).

4.1 Functional Requirements

4.1.1 Definition Used to Assign the Functional Requirements

Many formal definitions for functional requirements exist and while there is a “broad consensus about how to define the term ‘functional requirements’”(Glinz, 2007, p. 21), there is no stated, standardised, definitions. However, the taken meaning that this project will be using is that of Robertson & Robertson (1994, p. 111) “functional components that are easy for the user to recognise”. This means that the requirements state the working parts of the project which are definable by the user; in this case, the developers, smart watch users and me.

4.1.2 Table of Functional Requirements

With this in mind, below is figure 5, the table of functional requirements for this project.

Requirements	MoSCoW Scale	No.
A Library to be produced that improves gesture control on the pebble smartwatch.	M	1
Library to have at least 3 default movements	M	2
Library to support the gesture, twisting of the wrist	M	3
Library to support the gesture, shake	M	4
Library to support the gestures of directional movement	M	5

Library to support a circular movement	S	6
Library (or possibly and an application) to have the capability to output real-time data without the developer having to track the Accelerometer Data	S	7
Library to support a running motion	C	8
The Library should be able to operate at a minimum 25 samples of X,Y,Z data at 10Hz.	C	9
Ability for library to learn new gestures	W	10
Library/Application to produce snapshot data to aid developers	M	11
Application to monitor noise levels	M	12
Application to detect “Down”	C	13

Figure 5 - Table of Functional Requirements

4.1.3 Justification and Analysis of Functional Requirements

Figure 5 states the functional requirements this project **must**, **should**, **could** and **won’t** implement, in order to meet the aims and objectives (chapter 1.1).

The library **must** requirement’s contains all the gestures that are necessary to complete the first objective, that “easy to do” gestures are implemented. However, it may seem questionable that requirement 2 stated that the library must have 3 default gestures, whilst already requiring more three gestures to be supported. The key word here is default. If a user of the library wanted to use other gestures, these gestures are input by the user. If there is no input from a user, the default gestures are implemented.

The other 2 gestures that are suggested in the figure 4, 6 and 8, are in the **could** and **should** brackets, because the library’s main aim is to recognise gestures, not generate them for people to use. However, the only **won’t** requirement states an “ability to learn gestures.” This is because it is beyond the scope of this project in time feasibility, but is a very likely future improvement. If this was implemented, both 6 and 8 would then be covered.

Requirement 9 would seem strange to only have as a **could,** due to the accuracy needed by the library in order to recognise gestures. The justification of this is that, without development testing in the prototyping phase, the optimal measurement settings are unknown. This seems sensible from the research already completed, but maybe unnecessary, hence the **could** status.

All of these functional requirements can be easily tested, due to their true/false or boolean nature. I believe they are accurate in describing the aim and objects (chapter 1.1) and are within the constraints (chapter 1.3) of this project.

4.2 Non-Functional Requirements

4.2.1 Definition Used Assign the Non-Functional Requirement’s

Much like, and even more so than, the functional requirement’s (chapter 4.1), many definitions exist that describe the non-functional requirement’s. However, unlike the functional requirements, there is “no consensus about the concepts … because terms are used without a definition or a clarifying example. ”(Glinz, 2007, p. 21).

On the other hand, if a general understanding of the concept does not exist, why do so many projects and software lifecycles state them? An interesting question with no clear answer. My understanding of the definition is well covered by Davis (1993, p. 307-340). It is the required overall attributes of the system, including portability, reliability, efficiency, human engineering, testability, understandability, and modifiability. The non-functional requirements in this project have been made with my personal understanding of the above reference as well as the overall interpretation of Glinz (2007, p. 22), that each attribute stated in the definitions can have both a broad and narrow meaning.

My personal understanding of the overall concept is that all requirements, outside of the functional aspects, can be non-functional attributes of the project.

4.2.2 Table of Non-Function Requirements

Below is figure 6; the table of this projects non-functional requirements.

Requirements	MoSCoW Scale	No.
All coding in JavaScript/HTML	M	1
Comments in source code to Support Library	M	2
Testing to take place on a default gesture, at a target of 80% accuracy	M	3
Project version tracked via GitHub	M	4
Any data collected is within the Data Protection Act	M	5
Ethical Checklist is not violated during development	M	6
Stay below 7/8's of allocated memory	M	7
Applications should be able to recover from errors without crashing the Pebble Application	M	8
Each original application should not exceed the Pebble Memory limits	M	9
The use of all Axis in the testing of a default gesture	S	10
Official Documentation to Support Library	S	11
Create Icons for applications and Library	S	12
Release application to pebble store	S	13
Over a period of testing, False positives should occur no more than 2 in 10 measurements	S	14
Library should be Open Source upon completion of project	S	15
Code written should adhere to common JavaScript practice	S	16
From data gathered to gesture detection, no more than 2 seconds should occur	S	17
The library should take up no more than half the allocated space of memory per application	S	18
Applications should be able to recover from errors without crashing the Phone application	S	19
The library should leave to door open for improvement with more default gestures	C	20
Each application should be able to run indefinitely without crashing the watch. (Measurements should not be cached for longer than needed)	C	21
A website that can handle application settings	C	22
The library can be ported to other JavaScript platforms	W	23

Figure 6 - Table of Non-Functional Requirements

4.2.3 Justification and Analysis of Non-Functional Requirements

Figure 6 states the Non-Functional requirements this project **must**, **should**, **could** and **won't** implement, in order to meet the aims and objectives (chapter 3.1).

Starting with the **must** ranking, all have been seen as critical to my aims being reached as well as the general maintainability and usability of my project. Numbers 1, 2 and 4 reflect the overall coding and standard practice of coding. The design justification of the choice of JavaScript can be looked at in chapter 6.3.1- Choosing the Language. Requirement 2; the commenting of code is a constant requirement of all development projects in order to

achieve good maintainability, but the choice to use version control is not necessarily an implied convention.

GitHub, the version control software in use, is a personal choice due to my familiarity with it. Its advantages are that it has the ability to create private and public branches for the different versions of the library, for example a release and development version. It is also suitable for this project as I am the only developer and the service caters well for single developer and small projects. In opposition to these reasons, although GitHub does have the ability for larger, multi-developer projects, I would not consider using the free version due to for a larger project due to various limitations. I would switch to the paid version, if this were to be continued into a larger project in the future.

3, 7, 8 and 9 are the **must** meet testing requirements for the library and each application produced. The Pebble, my chosen platform (see more about this in chapter 6.1.2 - Choosing the Platform), has memory limitations due to its small nature; 7, 8 and 9 test that this is not exceeded at any point. Requirement 3 targets the algorithm and the gestures themselves, giving a tangible goal percentage to meet. Although 80% is a good success rate, I hope to be above this.

The Data protection act and Ethical Checklist (see Appendix B), requirement's 5 and 6, can never not be met. This is law and, although implied, should always be mentioned if nothing else to remind myself there are rules to be followed in development.

In the **should** ranking, 10, 14, 17, 18 and 19 are the testing statements here. These are **should**, rather than **must,** as they don't necessarily affect the outcome if not met. Requirement 10 asks for all 3 of the axis to be used when testing to ensure consistency.

Requirement 14 covers the non-default gestures which aren't compulsory to the objectives. 19 is a **should,** as this may be out of my control to fix. An error with the Pebble Phone Application may be the cause of such errors. 17's time frame is a usability aim; the gestures are still detected, this only seeks to improve the usability of the applications that may implement the library this project produces. Requirement 18 is also about the usability of the library. If the library uses over half of the allocated memory for an application it would dramatically reduce the availability of memory for a developers own application.

I plan to release the produced library and any apps I produce 11, 12, 13 and 15 cover this option, **should** they be of the quality needed or suitable for realse. 16 relates to requirement 15. I plan to release it in Open Source, following common JavaScript practices would improve the maintainability and readability of the library. However, JavaScript is a very versatile language and what this project is attempting to implement, to my knowledge, has not been completed before in this language. Henceforth why unique challenges may cause a failure of the 16th requirement and **should** try to be followed, rather than **must** be followed.

Three **could,** non-functional, requirements have been stated. Requirement 21 is a testing condition. This requirement hopes the achieve longevity in the running of applications to ensure quality to the users. 20 and 22 are of **could** ranking because they may not be

needed during development but should be considered in the process of the artefacts construction.

This library is being tested only on the pebble and only with this platform in mind. However the final requirement, of **won't** category, is that of porting to other JavaScript platforms. This is seen as a future improvement. See more about this in the chapter 8 the conclusion.

4.3 Conclusion of Requirements

As stated throughout the chapter, although a lot of the requirements may not be of the **must** category and critical to development, many will still be completed by the artefacts produced.

Each application will be tried against its applicable requirements and the reports of these comparisons will be available at the end of each implementation section, when talking about the developed artefact.

All requirements are of a quantifiable nature, whether that is of boolean value or numerical terms and this will also be reflected in the comparison reports.

5 The Library - Gebble.js

This chapter is a description of the design and implementation of the library which this project has constructed in order to meet its aims and objectives. This section does not state the development stages before the construction of the library and the prototypes that lead to it. This is outlined in chapter 8.

This chapter will report on the meeting of the requirement's outlined in chapter 6 and a small reflection of how it went from my point of view. An overall conclusion can be found in chapter 9.

5.1 Design

The library had the largest design aspect of the artefact. With the library inherently only going to be used by developers and applications, the interaction between the user and my library would have to be designed with a data flow diagram rather than with various diagrams of a proposed interface. But this had to tie in with the concept used to implement the algorithm. Seeing that the algorithm would affect the interaction with the user, this was designed first, then the data flow diagram afterwards. At this stage of the project, this concluded my design. Nevertheless, the implementation chapter 5.3 contains improvements on this design which were overlooked at this stage.

5.1.1 The Theory behind the Algorithm

The design of the algorithm used is based, partly, on a proposed method of gesture detection in a paper titled "Gesture Recognition with a 3-D Accelerometer,"(Wu, Pan, Zhang, Qi & Li, 2009, p.25-38), discussed in the literature review. Figure 7, is a diagram representing the element of the article I took inspiration from.

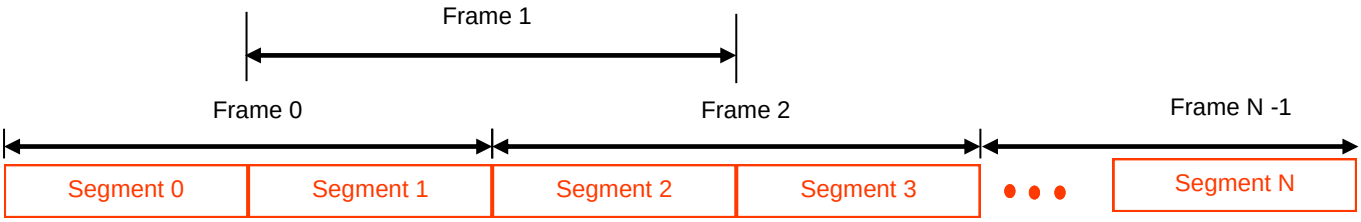


Figure 7 – This is a visual aid to the explained algorithm in this section. It is a graphical representation of the array form that is used to compare the samples supplied by the Pebble's accelerometer, to the tolerances supplied by the application using the library. Based on the Diagram provided by Wu et al (2009, p.27)

The Pebble generates a set amount of samples in arrays, at a set interval, across a second. This variable is specified in my requirements as 25 samples at 10Hz. This means at a rate of 10Hz, arrays containing 25 samples of the accelerometer at a certain time are being posted in an event. These variables were chosen as a result of testing in the first applications. They can be located in Appendix E.

This array will then be manipulated into the above format. Each single sample from the event is a segment; 2 segments make up one frame; in this example 24 frames then make up the array to be passed into the detection algorithm. The library now has the accelerometer data in the format above.

The array is deployed in the format above as it makes a segment to segment comparison easier to evaluate. If not in this format, the cost of going back and forth within the array to compare the gesture would have been less efficient. This is important because the evaluation of a posted accelerometer event must be completed before the next event is posted to stop queuing and delayed recognition, which affects usability of an application.

The next stage is segment comparison. By taking one axis value away from the corresponding axis in the conjoined segment, in an absolute form, you end up with a positive difference. This difference, in G-force, is what the library uses at its comparison stage.

Gestures are a set of differences across a number of frames. They have an upper and lower tolerance, an improvement made in prototype 2, chapter 6.3, that a frame must fall in to be identified in the detection algorithm. The longer the gesture (in number of frames) the more accurate the detection, since more comparisons will have to be made in order to evaluate each frame as true and therefore the gesture as a whole. Priorities are then assigned to each gesture that is compared in the same array. For example, if you wish to remove violent shaking from the equation, this will be a priority of 1. The rest of the gestures will then not activate when the Pebble is being miss used.

Moving on to the actual comparison of the frames to gestures, firstly, 2 checks will be made; An initial check of the time field and a check to see if the vibrator was active during that frame. If another gesture has been completed, the library allows for the user to return to their natural position before another gesture is then detected, this is the check of the time field of the first segment in each frame. This will be made available for developers to configure themselves. A helpful variable is passed in the object of each segment, from the pebble, “vib”. If this is true, the frame is discarded as well as that current gesture. The vibrator, in the watch, affects the accelerometer values.

After both these checks are completed, all the lower bound and upper bound comparisons are made to each x, y and z value. However, if one comparison is not true, that frame will be instantly discarded, meaning no other evaluations take place. This improves the efficiency of the algorithm and helps working within the constraints applied by the pebble, processing and memory power. Subsequently, if all checks are passed for each frame, a gesture is detected. A true value and the detection priority will be passed to the application and the algorithm put on a timeout until the time field value has been reached.

On the next Page is a flow chart (figure 8) that represents the comparison algorithm as explained above, but this flow chart only assumes that 1 gesture is being detected. If this were not the case, the flow chart would become hard to interpret.

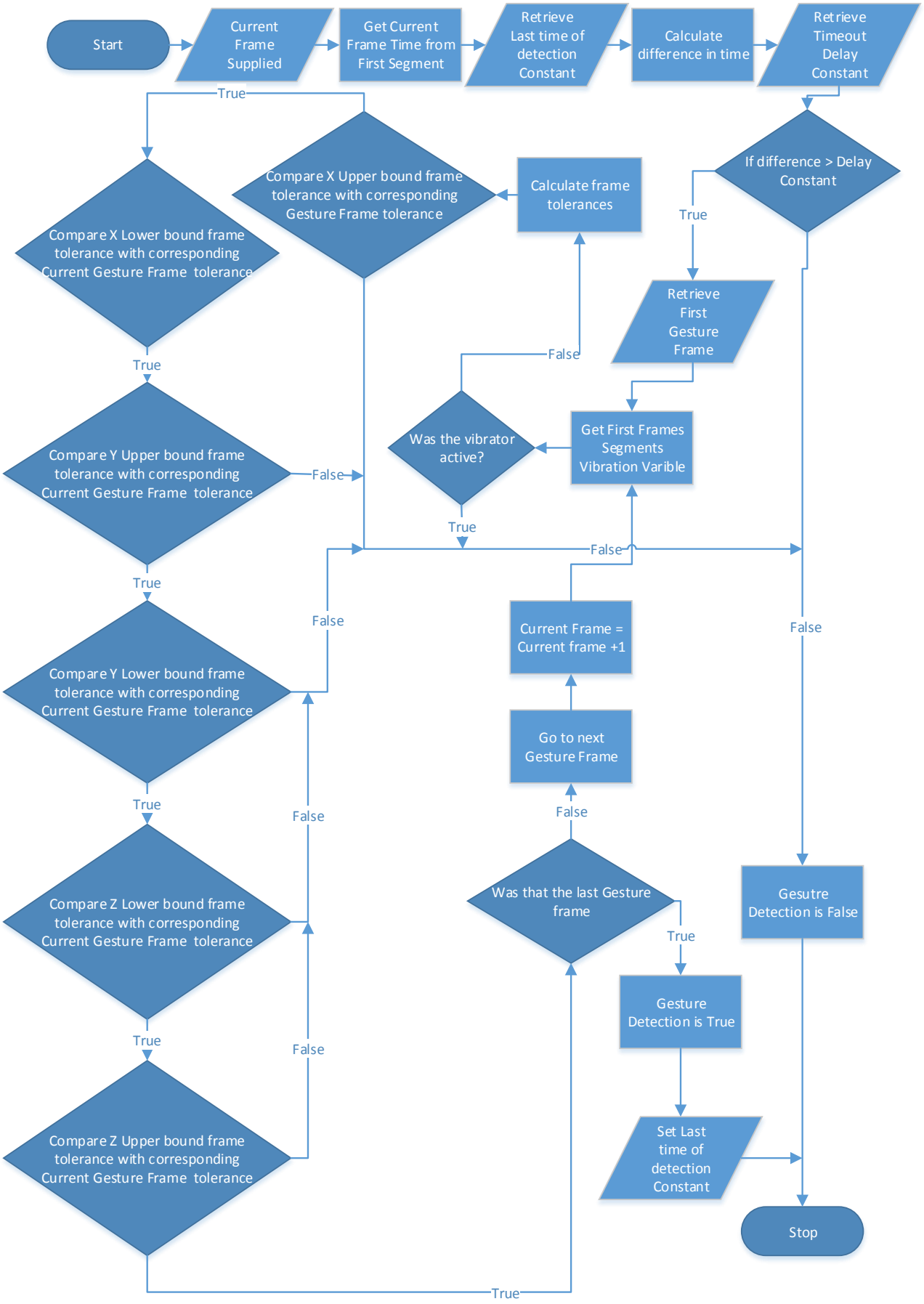


Figure 8 - This flow chart represents the comparison algorithm as explained above. We are assuming there is only one gesture to be detected for simplicity in the diagram.

This concludes the explanation of the algorithm. Whilst this design was implemented, an oversight was found during the implementation which had to be fixed. An explanation of this is given in 6.3.2.

5.1.2 Data Flow Diagram

Here lies the Data Flow Diagrams. These diagrams are a graphical envisagement of how Gebble.js is going to work.

5.1.2.1 Context Diagram

This diagram represents the top level communication between Gebble.js, the application being run on the Pebble and its own accelerometer. This design is from the point of view of Gebble.js, hence why the Pebble application and accelerometer are external entities. The boolean value returned will be true if a gesture has been recognised and false if not. The number of the recognised gesture will also be returned so the user knows what gesture has been detected.

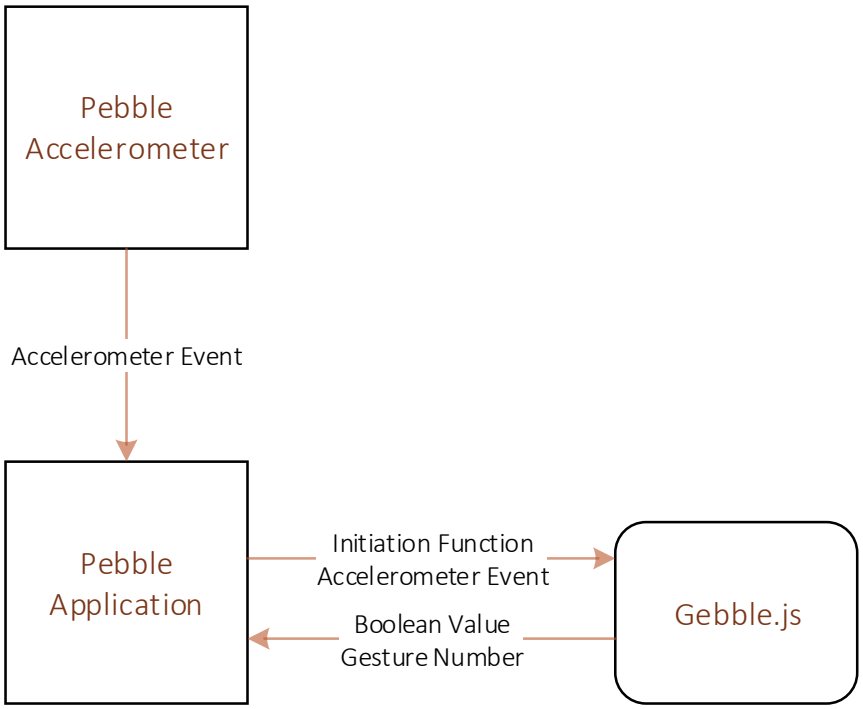


Figure 9 - This is the Context Diagram for the Gebble.js Library

5.1.2.2 Level 1 – Gebble.js

This diagram represents the envisaged inner workings of Gebble.js. The inputs and outputs remain the same as the context diagram (figure 9), but can be sourced from different places in the program. For example, “False” can be issued in the “Timeout Checking Procedure” and “Iterate over frame Array matching to gesture” processes.

The three constants set by the “Init function” are variable to the user. “Debug” allows console logs to be printed across the system. “Gestures” is and array containing the gesture the application if to detect and “Timeout Delay” takes the time that should be taken

in-between detection of gestures. These all will have a default value, should init not be used, these are Debug = true, Gestures = shake, X movement, Y movement and Z movement and Timeout Delay = 3 seconds. “Time of last Positive Frame” is set to the first of the last section of frames that were evaluated to be a gesture.

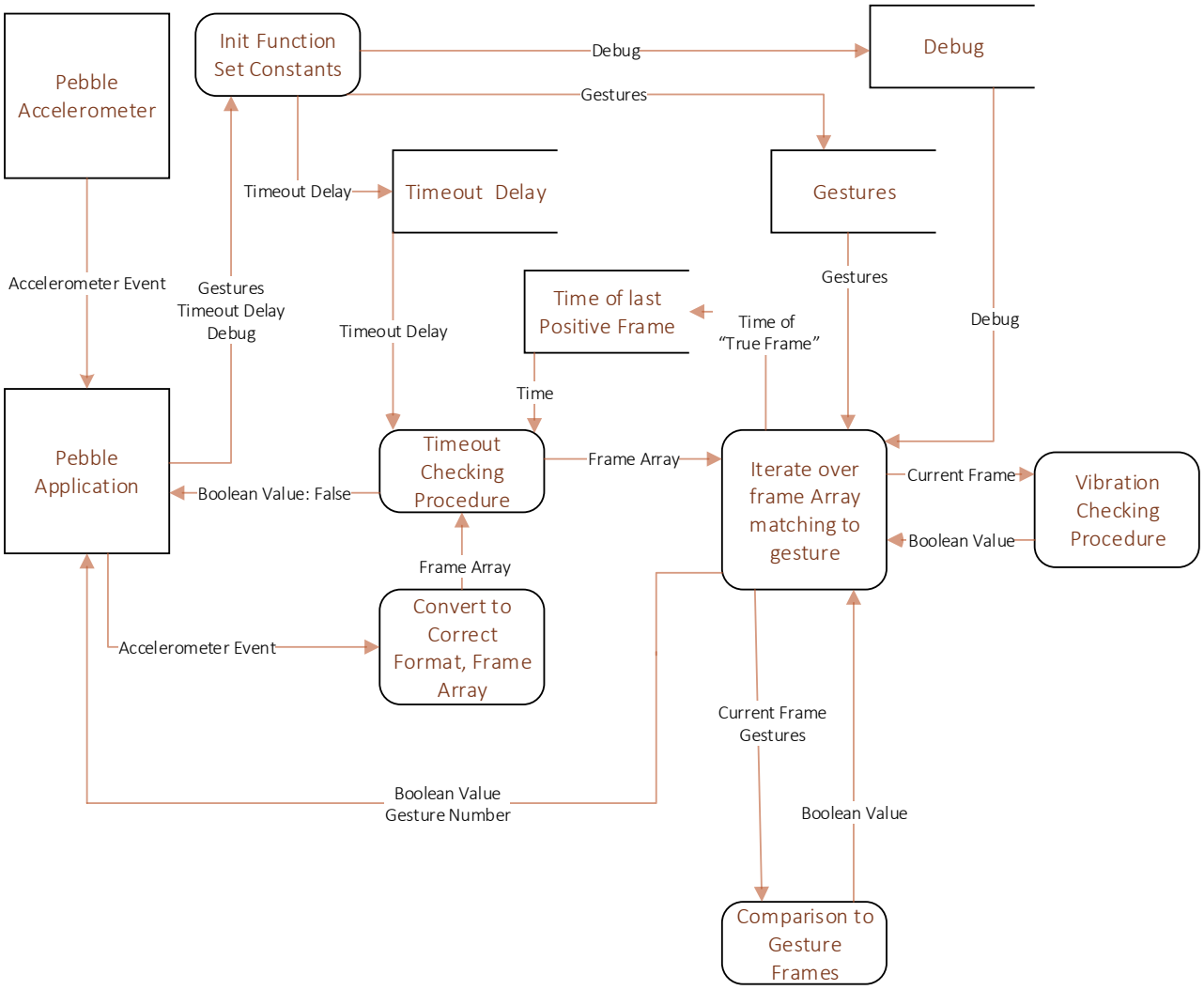


Figure 10 - This shows the processed to be involved in the Gebble.js library

This concludes my design section for the library. However, as mentioned in the introduction, this design was by no means concrete and an oversight was found. This is talked about in 5.3.2.

5.2 Implementation

The implementation stage of my library is an ongoing process, and continued development will be carried out because of the potential it holds outside of the time frame of this project. The headings below outline the stages of implementation that were completed during this project, the conclusion, chapter 7, talks about future development.

5.2.1 Evolution from the Prototypes

Chapter 3 outlines the methodology used to complete the deliverables of this project. The evolutionary process in use meant that I was able to use the 2nd prototype,(6.3) as a basis for the library. In the 2nd prototype the algorithm and most of the functions needed had been implemented but were integrated with the application. The first task was to then extract all the necessary functions, which the library would need to operate independently. These functions were extracted into a separate JavaScript file, which was to become Gebble.js.

I now needed to implement the interaction between the library and the user. 5.1.2 shows a single method would be the main interaction between the application and the library. This would take the event supplied by the application and then run the Gebble.js with its data. However, the option remained for the developer to use the base functions, should they want to. Implementation was started with making the file a module and exporting the functions to be used, but this proved very difficult due to an error with the debugging platform, Cloud Pebble. I go into more detail about cloud pebble in 6.2.1 but this is an online platform with a real time JavaScript debugger.

This debugger was showing the line “module.exports” to not be supported. Pebble appeared to support this function in its documentation and the open source code on Github. The next logical step was that my implementation was wrong but the code was tested and worked fine on the other platforms.

Out of interest, the project was built to see what error would be output upon compilation, but everything worked. The error was with cloud pebble’s internal debugger. In fact, there had never been a problem at all. However, all the effort of testing did not prove futile as much was learnt about pebbles internal workings. A bug report was submitted, but I have had no response from this report. After this submission, I moved onto the next evolution, the implementation of initialisation variables.

These variables were, debug, delay and gestures. All had defaults already assigned should they not be needed. Debug, being set to true, allowed output in the form of console logs. These range from the output of the detected array to the priority of the gesture detected, this satisfies the privacy investigation from the literature review. Delay and Gesture are the same as explained in the design.

From this point, the library was completed to a good standard and was ready to be released. However, initial testing showed a part of the system had been overlooked and this needed to be fixed before progression into release could be made.

5.2.2 Frame Overlapping

It was noticed JavaScript produced an error which counted outside of the upper limit of the array. This did not happen to each iteration of the loop. In fact, whilst trying to debug the error, I observed that the error happened during detection of a gesture and not when the Pebble was stationary. The error also appeared to be random, with no correlation between

the number of loops completed and the number of errors produced. This led me to believe it was a problem within the gesture array.

After testing this theory, a problem could not be found within this array. Turning my focus to the loop itself, I discovered the nature of the problem, a large oversight in the design of my algorithm.

If a frame, in the input array, was a match to the first frame of the gesture, it moves onto the next frame for detection. Yet, if it was the last frame of the input array, whilst the outer loop knew not to continue, the gesture loop would. This would then lead to an out of bounds error within the frame array.

In order to fix it, a way to conjoin 2 arrays would have to be found; these being the previous and current frame array. This provided a problem of its own, at what point in the algorithm would a conjunction be made. The answer that first comes to mind is the end but this would mean Gebble.js have to look for the next event whilst still in the current one. A solution to this was not found that was efficient or maintainable, as it exploited a back door.

Now looking at the start of the array, if it was possible to cache the previous array and then join them at the start of the next frame array, it wouldn't need to wait for the next event inside of the loop. Instead, process the whole thing as one each time. The exception is the first iteration as there would be no previous array.

This solution worked but a condition was used to start the processing stage at the length of the largest gesture, minus 1, from the end of the array. For example, if the length of the longest gesture array was 4 and the length of a frame array was 24, it would start at position 21. Calculating the start position seemed like a waste processing power, something the Pebble does not have much of. This solution would also need a hardcoded frame array length as a global variable, something which is bad practice. Therefore, my final solution was developed.

The final solution would be to create an overlap between both arrays that was as long as the longest gesture array. Each time stepping along by one frame created a new array. The algorithm would then lower the upper limit to that arrays length minus the longest gestures length. The detection would then stop at frame 20, if the length was 24, allowing the last gesture to be detected if the longest gesture was of length 5. The next array would then contain frames 21 – 24 of the cached array and frame 0 of the current array to close the gap. Figure 11 explains this graphically. This completes my initial implementation for the scope of this project.

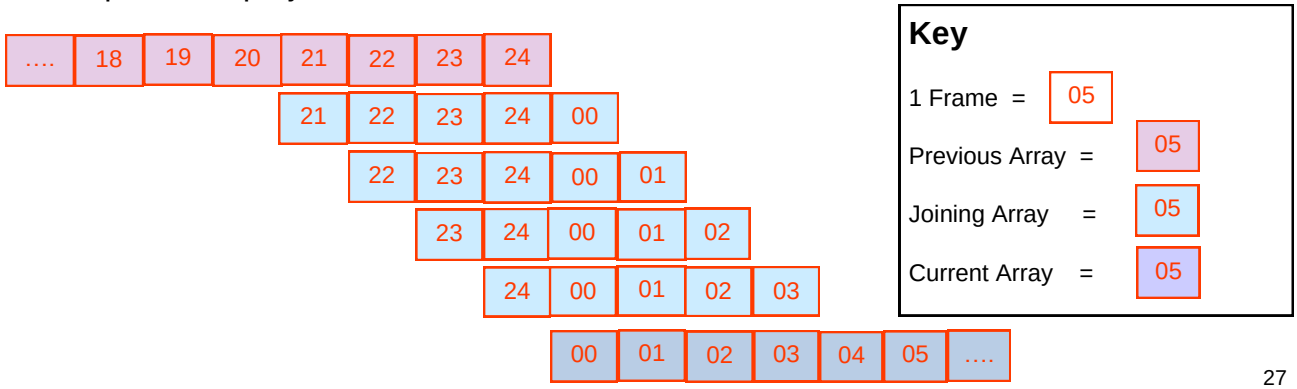


Figure 11 - The diagram represents the method used to join the previous and current frame arrays together.

5.3 Requirements Evaluation and Testing

Evaluating against the requirements requires different approaches for different requirements. Some will require testing as a means of verification and validation while others will only require and informed yes or no decision. However this does not mean that those requirements, of a boolean nature, should be taken with any less amount of investigation.

5.3.1 Functional Requirements Evaluation

Chapter 5.1.2 highlights the functional requirements of the project and numbers 1 through 10 represent the requirements, I felt at the start of this project, that this library should meet. However, over the duration of this project, I have come to realise some are more important than others to my main aim, improving the ease of use on the smart watch.

Requirement 1 can be evaluated by answering the question: Has Gebble.js, improved the ability to use gesture control on the Pebble smart watch? Simply, yes. However, as the creator of the library this is a biased opinion. From the point of view of a developer, before Gebble.js, there did not exist a way in which gesture control could be implemented in JavaScript, unless the developer went through the process of creating a module themselves. Something not all developers may be able to do. Gebble.js now exists and has therefore **improved** the availability of gesture control to at least the developers who cannot make one themselves.

Requirement 2 states 3 default movements should be implemented. This has been met. As stated in the implementation section, if an application does not set an array in the gestures field, the default gestures are used. These are a shake, X, Y and Z directional movement, one more than the quota.

Requirement's 3, 4, 5, 6 and 8 have become irrelevant, from a specific point of view, but some have still been covered. These requirements require the library to support a specific gesture or motion. From the point of view of the Gebble.js, this is not completed by itself but by the definition of gestures that a developer wishes to recognise in their application and then pass to Gebble.js. If these movements are included in Gebble.js, this is extra code that may not be used by a developer and therefore decreasing the amount of space available to the developer, for their own application. This is because the library, in its current form, comes as one package. However, directional movement and shaking is recognised since they are the default movements that demonstrate how the library works in its crudest form, whilst still being useful. This problem could be solved by creating different packages with customisation options, but this is beyond the scope of Gebble.js in this timescale of this project. This is a future improvement covered in the conclusion, chapter 7. From the point of view that all gestures are supported if you implement the right tolerances, all these requirements are met. But, currently only requirements 4 and 5 are met by default gestures.

Requirement 7 was met by an application in this project. Appendix E talks about this application.

Requirement 9 states a minimum operation specification for the Gebble.js. This is the rate at which Gebble.js currently operates, so the requirement has been met. However, during prototyping it was determined that this was an optimal rate at which to test detection. This has not been changed and although a developer defined gesture will work at a higher rate, the default gesture will have a lower accuracy. This is because more segments are produced by the event, with no increase in the frame number that the default gestures hold.

5.3.2 Non-Functional Requirement's Evaluation

Chapter 4.2.2 introduces the non-functional requirements of the project. Whilst most of these requirements are fulfilled by the boolean type yes or no answers, some have required a test plan. These test plans are available in Appendix C.

Requirement 1 states that all coding was to be completed using JavaScript and HTML, a relatively easy requirement to evaluate. The library and applications have all been written in JavaScript. As well as this, it has also been logged on GitHub and commented throughout, meeting requirement 2 and 4. 5 and 6 require me to follow the ethical checklist (Appendix B) and the Data Protection Act; to my knowledge, this particularly crucial requirement has been completed. For Requirement 11, the documentation for this library is in Appendix D as well as online at maxhunt.github.io. 15 requires the library to be open source, I have moved it to a public branch on my GitHub.

Requirement 16 states I should adhere to common JavaScript practice. I have, in my opinion, adhered to common coding practice, but this is dependent on the view and developing style of each user.

Requirement 18 was complicated to verify. This declares the maximum size of the complied library at 50% of available memory. On its own, as a single file, when built and sent to the watch, it takes up 19%-20.5% of the memory. However, if combined with an application, on average, it increased to size by about 0.5 - 1%; both results are below the threshold of 50%. The initial cost of building an application, for the watch, appears to be high, additions of files are then are relatively small. By making the reasonable assumption that the library will only ever be used in conjunction with an application, this means the overall memory cost is around 0.5 – 1%.

Requirement 20 relates to 15. By being open source, this requirement is met. The default gestures are obvious, as well as commented in the code, and could be easily added to. This is met. However, Requirement 23 is not. But as a **wont** requirement, this was inevitable. This is talked about in the conclusion, chapter 7, as a future improvement.

We move onto the non-functional requirements that are verified via test plans. The first, that applies to the library and needs a test plan is requirement 14. In order meet requirement 10, this need to be tested over all axis. Therefore I will be using the default gesture of a shake, as this implemented over all axis. The test plan, and description of it, can be seen in Appendix C - 1. The result of this was 30 passes to 2 fails, passing this requirement. However, the 2 fails were as a result of testing hard, random movements, easily construed as a shake. If these results are taken out of the test, 100% accuracy is reported. As this is also the default gesture, is completes requirement 3 as well.

Requirement 17 required more of improvisation with testing(test plan Appendix C – 2). This tested the time it takes for the detection of a gesture, set at a maximum 2 seconds. The solution was to use a colleague to start a stop watch at the end of gesture, and stop when the detection is achieved. Again all axes must be tested to comply with requirement 10. This requirement was achieved at an average of 0.733 seconds in average response time. All measurements taken were under the threshold and there were no miss detections or false positives.

Therefore all requirements, that are applicable to the library, have been completed with the exception of the **wont** criteria, but this is discussed in the conclusion as mentioned before.

5.4 Conclusion and Reflection of Gebble.js

The library has met all of my expectations, as well all of the requirements. I would therefore say it was successful in the achieving the projects aim; it improves the ease of use of smart watches, via gesture control.

Overall my view of this stage of development was that even though it was difficult at times to find motivation to complete Gebble.js, the end product not only meets the requirements but is in a maintainable and in usable state for someone to apply to their Pebble application.

6 Case Study Design and Development

This chapter outlines the initial design and development and both prototypes before Gebble.js.

6.1 Initial Design and Development

The overall aim of this part of the development process was make way for the prototypes that led to Gebble.js. Many design decisions were made before and during the process of this development stage. An explanation of these decisions and a discussion into the way the prototypes were implemented will be given.

6.1.1 Choosing the Platform

Whilst there are many smart watches currently in the private sector, I have chosen to develop on the Pebble Platform. I believe it was the correct choice for this project.

The drawback of this decision was that the Pebble SDK, software development kit, can only be used on a Linux or Mac system. Neither is reliably available for the duration of this project. This was overcome with the discovery of the Cloud Pebble service.

Cloud Pebble allows programming of the Pebble online. It also syncs directly to GitHub, the version control software in use. Simulation of the Pebble watch offline is another valuable feature, as well as an in house debugger JavaScript. With the discovery of this service and all the additional features, my initial constraint was overcome.

Figure 12 -Screen Shot of SnapShot Application

Once this issue was sorted, the Pebble was the clear leader in all areas including price, battery life, ease of development, ability to screenshot application (see figure 12) and the development community surrounding it. It should also be mentioned that at the time of choice, the Apple Watch, Moto 360 and Pebble Time had not been released.

6.1.2 Choosing the Language

As mentioned before, and defined in the requirements, JavaScript is the Language in which the project is to be developed. However, the Pebble also has the capability to install applications developed in C.

Although C is a very versatile language and I had already acquired skills using it, I believed that the JavaScript option would not only enhance what I gained from this project educational, but also meant that the code maybe transferable to other JavaScript platforms in the future, another requirement.

The first applications to be developed are in Appendix E. They do not bring any insight into the projects goal and are more personal development rather than good implementation.

6.1.3 Realisation of Imperfections – The Noise Monitor

After the development of the Real Time Application (Appendix E) and during the testing of the requirements, a bug was reviled which was not expected and hadn't been considered until this point, noise.

"Noise is a random and persistent disturbance that obscures or reduces the clarity of a signal or measurement". – ("Noise", 2015).

Non-Functional Requirement 23 states that *"Each application should be able to run indefinitely without crashing the watch. (Measurements should not be cached for longer than needed),"* and in order to test this, the watch was left with the application open and laid on a table, to which there was fluctuation in the measurements, given by the axis, when stationary.

Figure 13 - Screen Shot of Noise Monitoring Application

If implemented gestures were accurate to the real measurement, noise may then influence this and cause a false detection or no detection at all. Therefore, this noise application was built.

The design was to take the accelerometer data and then implant it into an array. A global variable would determine the number of samples it would take before calculations were made. The outputted data would be the Noise values.

This would be calculated by collating all the samples and sorting the array, from lowest to highest. The last and first sample are taken and subtracted from each other, absolutely. This is the noise level for the concerned axis. The technique is repeated for the other axis. As well as the Noise level, the minimum, maximum and average values are posted to the application as they were useful to reference in testing and to verify the values were being calculated correctly.

One of the main challenges was a GUI problem. Multiple instances of the measurement window meant that the counter would increase by the number of times you had used the back button to exit that window and then started the measurement again. For example, if the user had exited the window once, after receiving measurements and started it again, the counter would then increase by 2. This created problems, with memory, but still did

provide accurate measurements. It also meant that if the application was closed, whilst the samples were being taken, the application would continue behind the scenes.

The solution to this problem was to introduce a deadlock around the counter, the critical section. Once one instance completed this section, it closes and the back button is press is mimicked, the stacks are wiped and therefore nothing else enters the critical section until the application is started again.

At this point the functionality of the application worked. Although an informal application, I decided to meet the Non-Function Requirement, Number 13, a professional finish. To me it was obvious how the application worked and I knew that it should be placed on a surface upon starting the application; a user would have no knowledge of this without instruction and a timer to give them the opportunity to get it to a stable surface. This was implemented, but with a problem.

Due to JavaScript's a-synchronous nature, a problem arose. Initially the implemented timeout function acted as a break point. But, this would be skipped and a handled later when the application, voiding the timeout. After some debugging, the implementation of the function had been misunderstood and it should encase the process to be held up rather than act as a break point. Additional benefits also came of this; the application was able post to the window every second. Thus creating a very user friendly countdown and leading to a good finish on the application.

I stated in chapter 3, that I foresaw some movement in the requirements. This is that case. Functional Requirement 12 was introduced as a result of this problem. An application **must** be created to monitor and therefore calibrate the noise levels. Figure 14 shows the results of the testing that took place, over 250 samples. With the exception of the 5th Z axis test, the noise levels, on average are 19.4 Hz for the X, 22.4 Hz for the Y and 21.6 Hz for the Z axis. The accelerometer is highly sensitive so a slight vibration of the test surface may have caused the erroneous data. A buffer of 21 Hz should now be used in gestures.

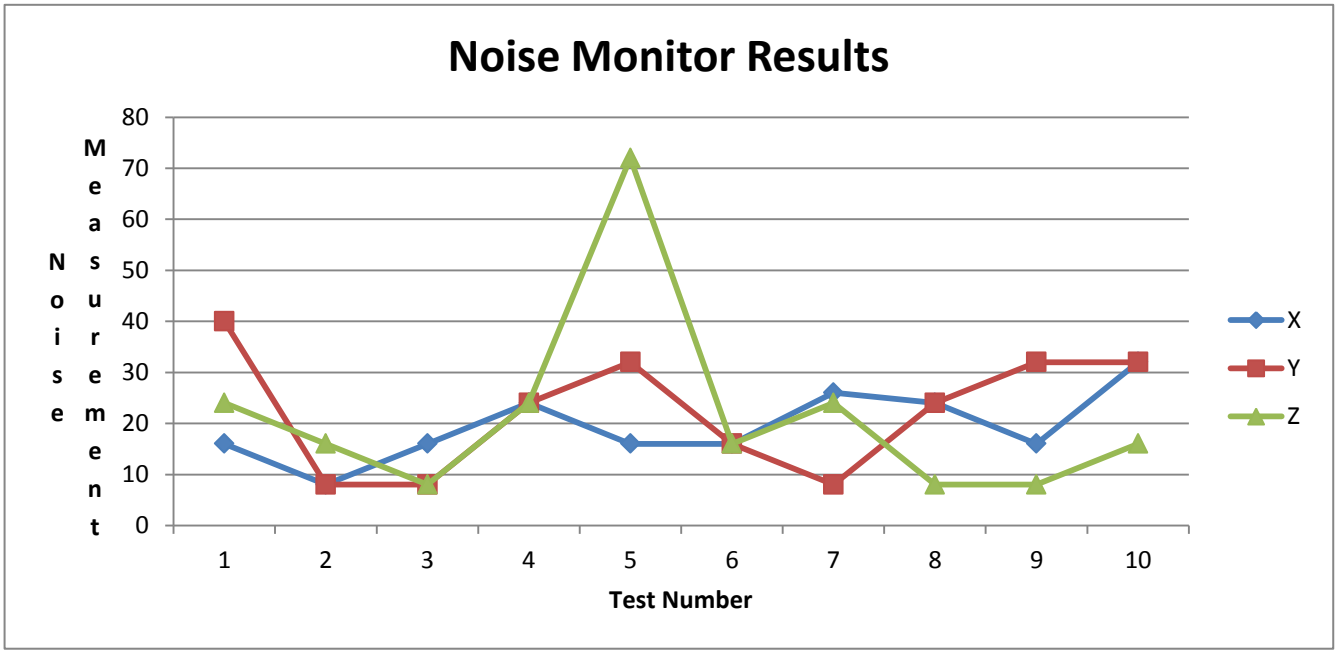


Figure 14 - Graph of Noise Level Testing

After discussion of the Non-Functional requirement 13, I have also met Non-Functional requirements: 1, 4, 5, 6, 7, 8, 9, 12, 13, 16, and 21. Functional requirements completed: 12. Non-functional requirement 7 asks to adhere to memory usage percentage of 7/8 or 87.5%. This apps usage was calculated to 20.5%, at required level. Number 12 requires an icon to pass this can be seen in Figure 13, next to the applications name.

It may also be noted that this application wasn't in the project plan, due to its unforeseen nature. At this point, there was already a slip in the monthly milestones. Chapter 4.3's gantt chart shows that this it should have been late November. Due to the problem encountered, this now had slipped to late December; prototype 1 therefore began implementation in January.

6.2 Prototype 1 – Wrist Twist

Wrist Twist was the first try at gesture detection attempted in the artefact. After the initial exploratory applications enough experience, with the Pebble.js API and JavaScript in generally, had been gained to start the design of this application.

6.2.1 Design

Whilst the design of the gesture detection algorithm is reflected by in chapter 5, 5.2.1 specifically, that design is a much higher up the evolutionary chain. At this stage, the discovery of a method of detection had only just been made.

With this is mind, figure 15 is a very rudimentary design of the algorithm to be implemented as a flowchart.

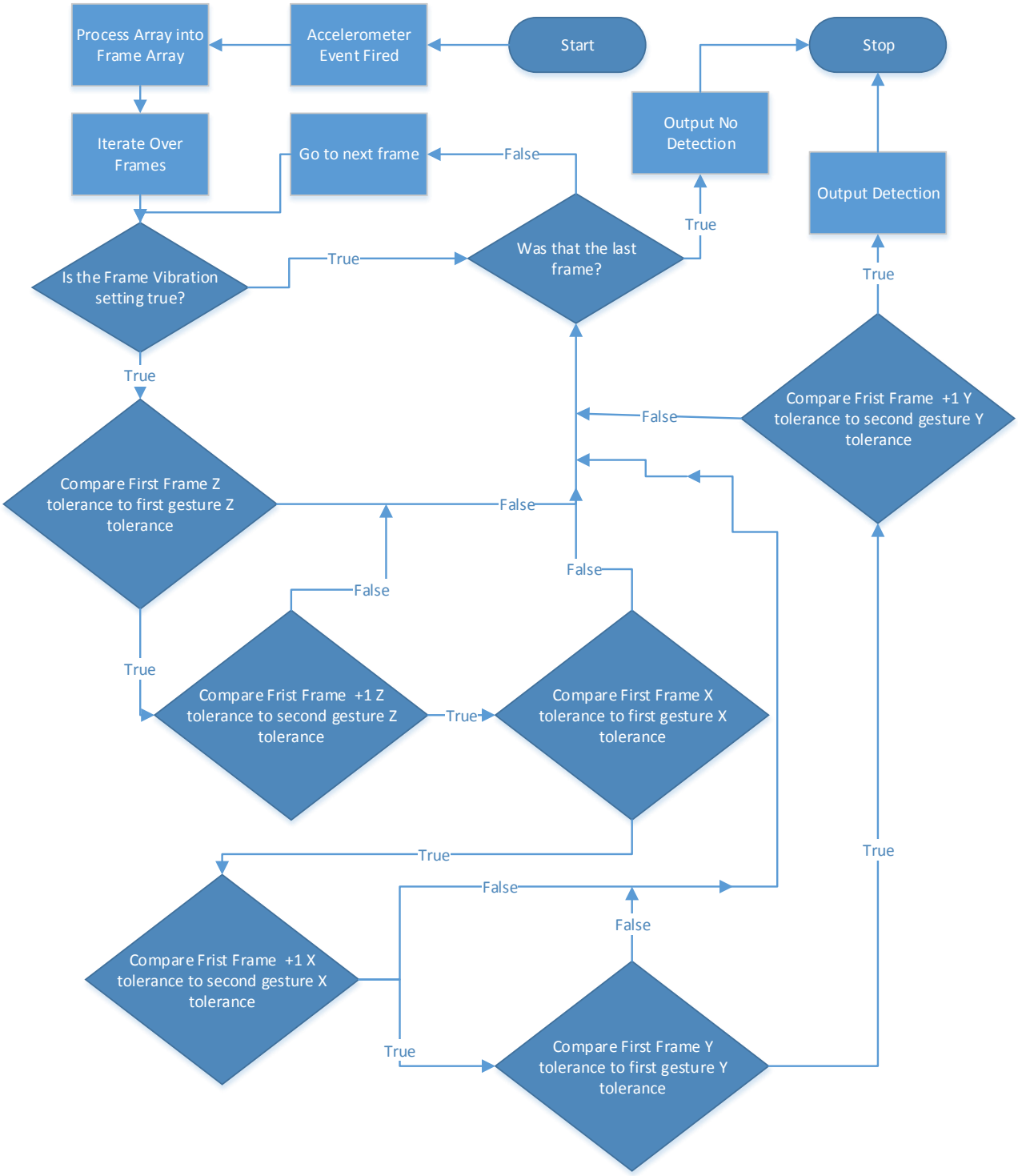


Figure 15 - Design of the algorithm used in the Twist Wrist Application

As this application was only ever going to have one gesture, there was no iteration across the gesture, instead it was hardcoded. The gesture contained only 2 frames hence why there are only 2 comparisons of each tolerance. These tolerances are upper bound only. Furthermore, this design is also missing the timeout Delay check, this was a development made in prototype 2.

However, it does prove an initial understanding of what had to be programmed in order to successfully detect a gesture. It also contained the manipulation of the accelerometer event, which converted the array into the form needed for comparison.

Unlike Gebble.js this is an application and has a graphical User interface. This interface's simple design showed instructions of how to use the application and the application name on the main screen. Upon the select button being pressed, the recognition would start and a counter initialised that would show the amount of twists detected. Figure 16 is a plan of this GUI.

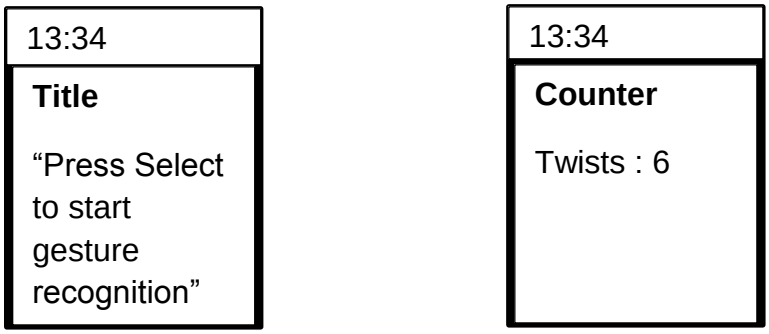


Figure 16 - Graphical User Interface designs for Wrist Twist

This concluded the design phase of the application.

6.2.2 Implementation

With plenty of knowledge from previous efforts and more energy put into the design stage, the implementation of this prototype was fairly straight forward.

The algorithm was implemented with ease. The design broke it down into a single loop and various if statements. However, it was the GUI programming that proved difficult to step up. Each piece of text in the GUI is an element that is inserted to the screen with a vectored position. But you cannot change the text of the object with one command. The original piece of text has to be removed, changed and then inserted again. If this process is not followed and the text is only changed, it writes over the top of itself. Once this was realised, the implementation of this application was completed.

Figure 17 - The Wrist Twist Application in the menu on the Pebble

In Conclusion, relative to everything else seen so far, this was the quickest implementation. it was due to the vast amount of discoveries made in the initial applications (Appendix E) and the amount of design produced. Figure 17 shows the

application finished, with its icon, in the pebble menu. Figure 19 and 20 show the end design of both the screens, in the application.

Figure 19 - The main screen of Wrist Twist

Figure 18 - Counter Screen in the Wrist Twist application

Although the implementation may have gone as well could have been hoped, the testing found various problems.

6.2.3 Requirements Evaluation and Testing

No requirements apply to this application as they all cover the finished library product and the separate applications produced at the start rather than the development process. But by applying the tests throughout development it also highlighted problems that would have no otherwise have been found.

Allocated memory, non-functional requirement 7, was calculated to 19.1% well below the stated tolerance.

Accuracy, requirement 3, tests had a very surprising result, highlighted in figure 20. This was that the average detection was exactly 80%. This was not expected to pass at such an early stage. More testing was needed.

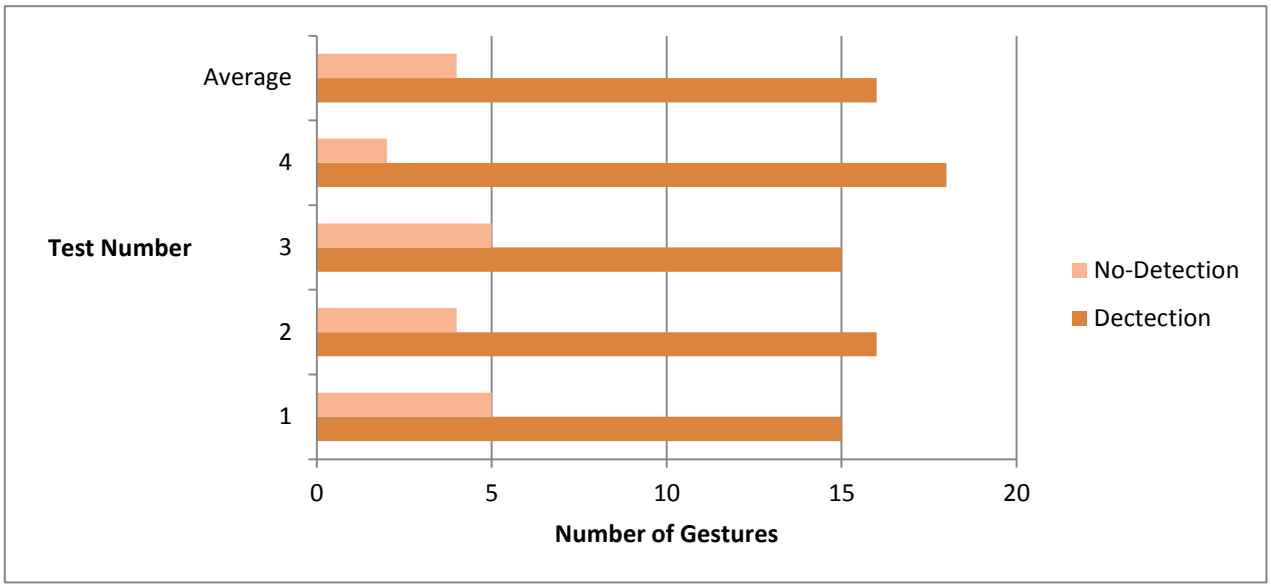


Figure 20 - Graph of the results from testing non-functional requirement 3 with prototype 1

Therefore I started testing requirement 14, and the answer became apparent. Figure 21 shows these results. The average here 36.5%, this contradicts to the overall result of requirement 3. There is a problem with the implementation.

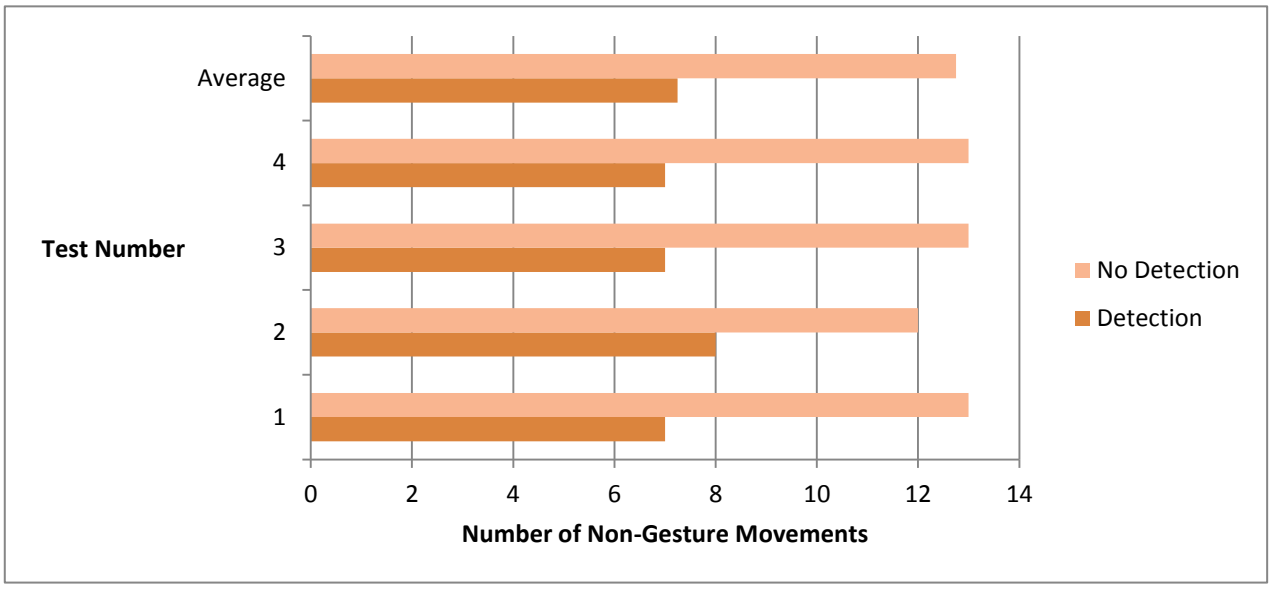


Figure 21 - Graph of the results from testing non-functional requirement 14 with prototype 1

After debugging the algorithm, the implementation seemed correct. The problem was the defined tolerances of the twisting action. They were too lenient. To solve this problem the allowance put in for the noise factor (chapter 6.1.3) would have to be reduced, as well as a lower limit introduced and a timeout function to allow the user to return to a natural position. This is exactly what happened in prototype 2. The detection speed is visibly random, sometimes over 5 seconds. In order to meet requirement 17, this will also have to be improved on in the next prototype.

6.2.4 Reflection of prototype

Overall, I think this prototype was a success. A working prototype has been produced with an algorithm, as well as a transformation function. There is room for improvement, but as this is the first prototype, that outcome was to be expected.

6.3 Prototype 2 – Gesture Counter

This project is using the evolutionary model of prototyping; this prototype is not starting from scratch, as the last one was. Gesture Counter is being built with prototype one as the foundation. This does not mean that there is no design aspect involved in this section.

The primary objective of this application was to fix the problems highlighted at the end of the previous chapter. These were:

- Add lower bound limits to the gesture detection
- Add a time delay factor
- Improve detection time of a gesture

The secondary objective was to alter the algorithm to allow for multiple gestures.

6.3.1 Design

Primary objectives 1 and 2 were a straight forward implementation evolution and need no design.

In order to achieve the 3rd primary objective, reading on the Java script language was needed and ways to improve efficiency were found.

The way loops were being implemented could be improved. Below is the generic loop from prototype 1.

```
for ( var i=0; i<array.length-1; i++){
     //dosomething
}
```

This is perfectly standard way to implement a loop, but an improvement can be made. By defining `array.length` in the loop header, each time a loop in made, this is recalculated; a very useful feature. However, if an array is not going to change in length over the loop, this is not needed. This can therefore be defined outside of the loop, but even this is not the best solution. If the loop is not needed, time has been spent defining a variable that isn't used. Therefore the solution below is best.

```
for ( var i=0, arrayLength = array.length-1 ; i<arrayLength; i++){
     //dosomething
}
```

By defining it in the loop header, all problems are avoided. This is something that I was unaware of.

Another coding improvement is within the comparison method I was using. The subject is nested "if" statements. These could all be brought onto one line, instead of consecutively

using logically operations; this included the new lower bound comparisons. The tolerances should also be calculated in these statements meaning that tolerances are only calculated if needed.

The next part of design is the technique for the addition of multiple gesture recognition. The proposition is to introduce an array of gestures which is looped over. The position in this array will determine the priority of the gesture. For example, the gesture at position 0 has priority 1; the lower the number, the better the priority.

Concluding the design, this prototype, being an application rather than the library, will have a GUI aspect. The design has more counters for the different gestures.

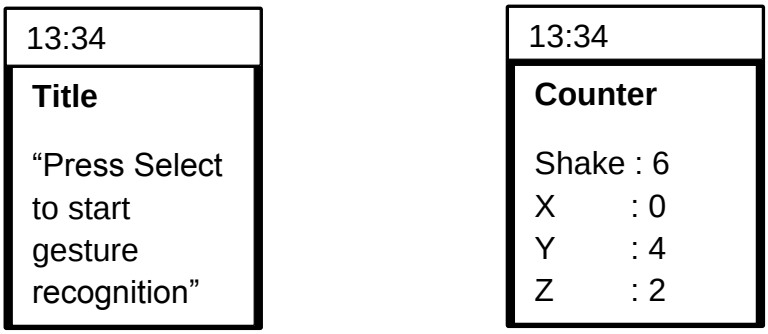


Figure 22 - Graphical User Interface design for prototype 2

6.3.2 Implementation

Continuing the theme from the last prototypes implementation, the thought put into the design made this stage a lot easier. However, problems were identified in preliminary testing.

A different approach was taken for this prototype and the GUI was completed first. It allowed the setup of counters and variables that would have to be supplied to. This was a top down development approach. Until now, a bottom up approach had been taken. Figures 23 and 24 show the implemented GUI.

Figure 24 - Main Screen of the Gesture Counter

Figure 23 - Counter Screen of Gesture Counter

The efficiency improvements were concentrated on the detection algorithm itself. As these were general coding improvements, rather than a new implementation, it was simple. Primary visual tests also concluded that speed up had been achieved; however, this was without the new improvements from the other primary objectives and the secondary objective.

Introducing the lower bound tolerances had little impact on the algorithm complexity as this was just another set of comparisons.

With the time frame implementation, there were initial problems with selecting the correct field in the array. This was down to a miscalculation rather than a problem with the coding. The effect this had on the algorithm was minimal. Nevertheless, there was difficulty with the calibration of these values.

In order to provide default gestures in the library, these gestures would have to be extensively tested. At this stage, I began to realise I was going of task and time would be better spent improving than prototype rather than introducing a whole host of gesture limits and time delays to be available for use. While this would be useful and improve the ease of use, the ultimate aim, implementing an efficient algorithm should be completed to a high standard first. Without that, these highly tested gestures would mean nothing to an application. Therefore, the time delay was left at 3 seconds and the default gestures that were implemented were a shake, X, Y and Z directional movement. Simple and easy to test gestures, that are still very useful to a developer.

Formerly, I had 4 gestures and an algorithm that could only detect one at a time. The secondary objective now had to be implemented.

By installing a loop within the main loop, discussed in prototype 1, this became possible. It also appeared to speed up the detection, even though more work was being undertaken. At the time, this was not understood completely. I thought may have be down to gestures being discarded a lot quicker because logical comparisons were being thrown away when false. This did help speed up, but the actual reason was discovered in testing of the requirements.

6.3.3 Requirements Evaluation and Testing

Due to the suspicions at the end of the previous section, the first piece of testing that was undertaken was in order to complete non-functional requirement 17, the time it takes to recognise a gesture. Initial tests, in the same format as the test plan available in Appendix E, put this average time below 0.2 seconds. This seemed too good to be true. After spending a time debugging, the problem was located. An incrementation on the gesture array had been missed meaning that the first frame was checked over and over again, rather than moving onto the next one. This also meant the time spent calibrating gestures was wasted, as this had been done while this error was present. Nevertheless, with this fixed and a recalibration of the gestures, the prototype was ready for testing. The visual time it took for detection also seemed a lot more realistic.

Non-functional requirement 17 was tested; the amount of memory usage by the application. This was a slight increase to 20.1% from 19.1% in prototype 1. It is under the

tolerance and therefore passes the requirement. The increase will be due to the increase of characters in the JavaScript file.

Non-functional requirement 3 targets the detection rate of a gesture. This was completed for all the calibrated gestures in this prototype. There is an extra field here as the gestures can also be miss-detected as another gesture. Figure 25, 26,27 and 28 show the results.

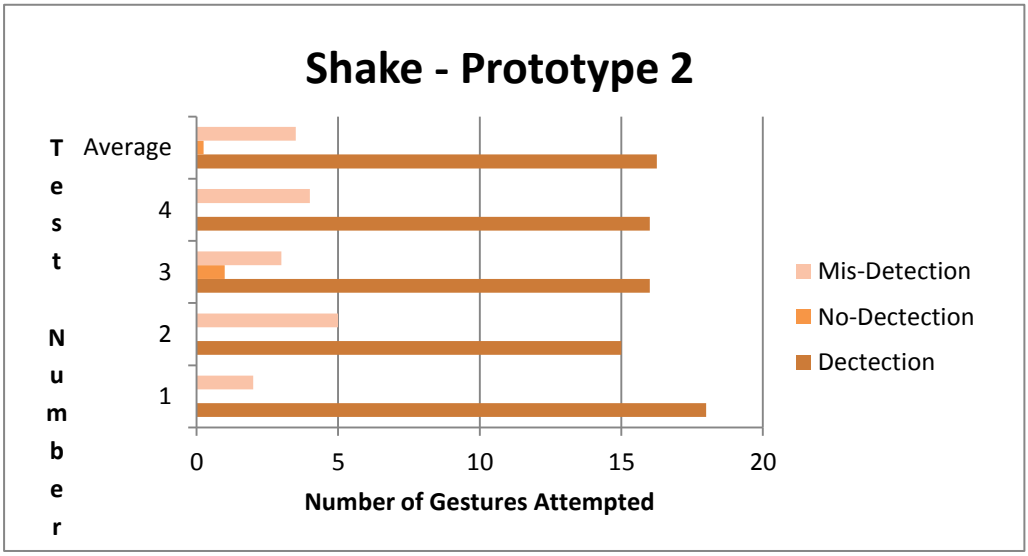


Figure 25 - Non-functional requirement 3 test results for the shake gesture

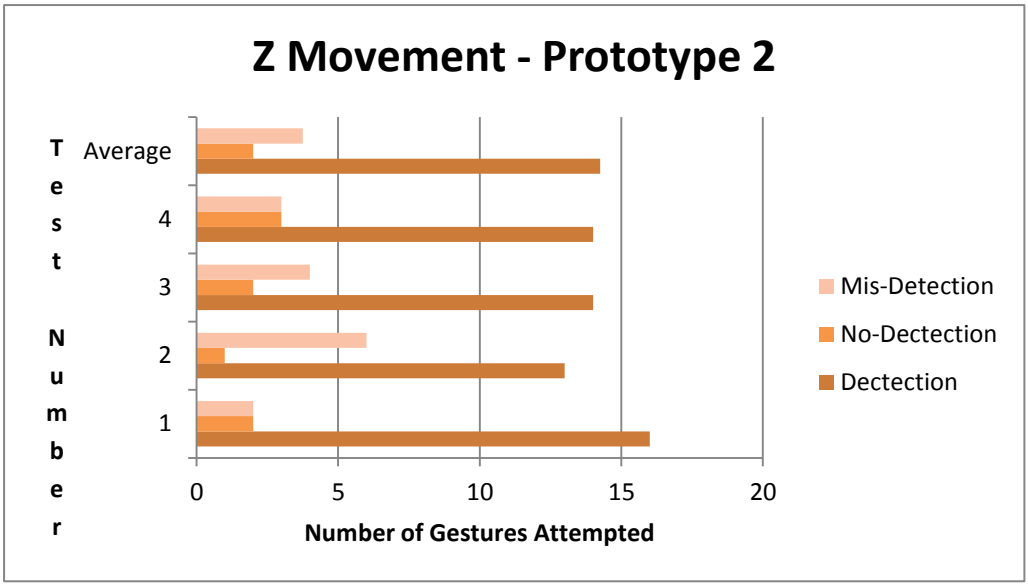


Figure 26- Non-functional requirement 3 test results for the Z movement gesture

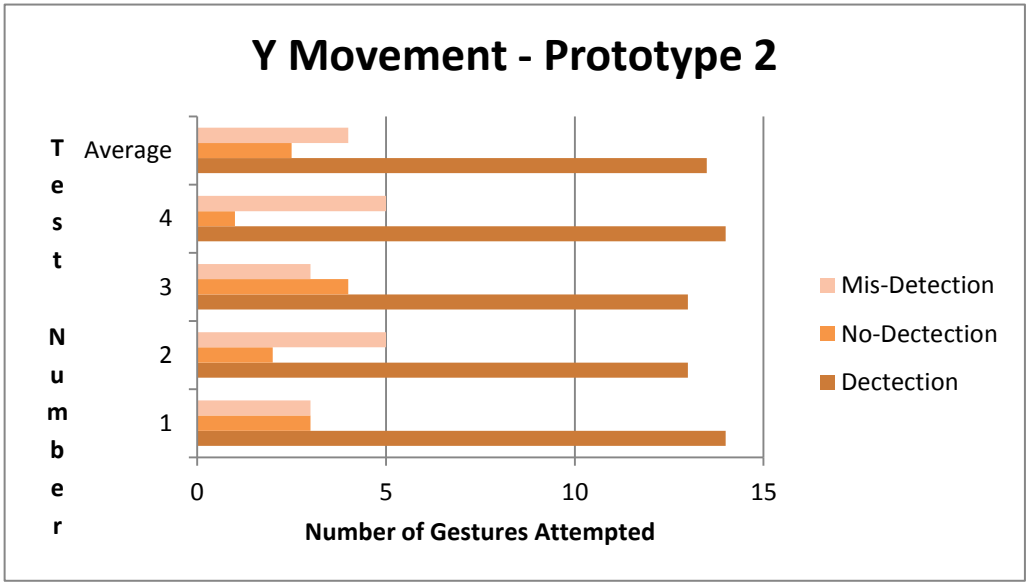


Figure 27 - Non-functional requirement 3 test results for the Y movement gesture

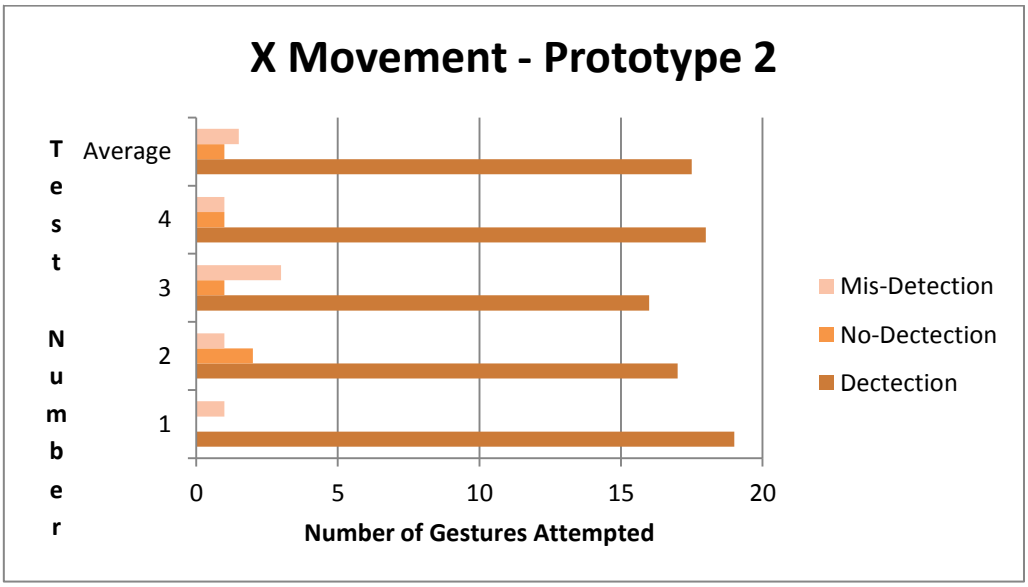


Figure 28 - Non-functional requirement 3 test results for the X movement gesture

Each test involved 20 movements of the gesture described. The result of the movement was then recorded. The shake gesture evaluated to 81.25%, Z to 71.25%, Y to 67.5% and X to 87.5%. Although, only 2 of these pass the required 80% threshold, for the 2nd prototype, they are sufficient. However, the reason that the Z and in particularly the Y measurements failed was still explored.

I believe this is because the shake gesture was more suited to X directional shakes. This meant that Z and Y movements were more often misconstrued as shakes than X movements, leading to the result that was found. In order to stop this happening, different shake gestures were introduced into Gebble.js, one for each of the axis.

Non-functional requirement 14's, false positives, result of testing was 18.75%. This is just below the threshold of 2 in 10. Figure 29 displays these results. This is a good result for prototype 2. This result also lines up with the other end of the spectrum, requirement 3. The overall accuracy of the algorithm and gestures has therefore come close to meeting

the requirements. The Gebble.js improved upon this result to meet these requirements and stated in chapter 5.

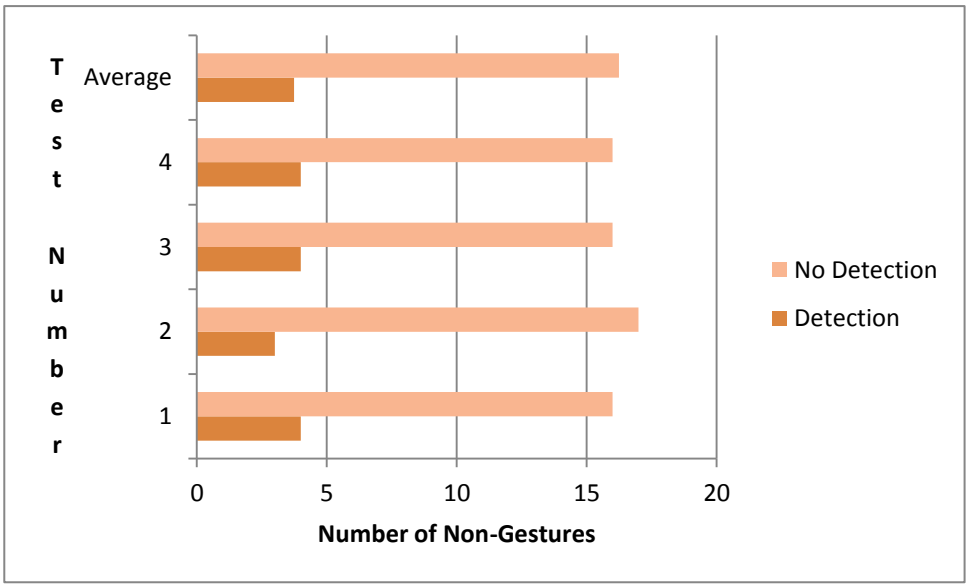


Figure 29 – Non-functional requirement 14 test results

Requirement 17 states that the detection of a gesture must be under 2 seconds. This requirement is the reason for the efficiency upgrades in this prototype. This averaged to 1.372 seconds, see figure 30.

Figure 30 - Non-functional requirement 17 test results

This is below the threshold, but spikes can be identified above. However, the efficiency has greatly improved from prototype one. Gebble.js improves this further after these initial findings as report in chapter 5.

In conclusion, great progress was made from prototype 1 to 2, in both testing and development.

At this point, I had regained time and was on target with the project management plan, chapter 3.3 (figure 4) . It was predicted that only 2 prototypes would be needed until a final version could be produced. This lead to a quick design and testing of the library, meaning I was also on target with my development plan concerning Gebble.js.

7 Conclusion

With all the development and testing completed, the end of the project has been reached. However, there are still topics that need concluding. This chapter speaks of my views on this project, including a personal review and future development. Also included is a comparison to the Aims and Objectives and an overlook from a project management point of view.

7.1 Overall Aim and Objective Completion

The overall aim was to “improve the ease of use on smart watch platforms, using JavaScript as the developing language.” This has certainly been completed through the implementation of gesture control. Here is an example of this.

The user is walking down the street listening to music; it is raining. A song comes on that the user does not want to listen to and wants to skip it. They now have to take their phone out and change song. However, its touch screen and the rain now hitting the screen is making it exceeding difficult to change song. The Gebble.js could fix this problem by allowing the creation of an application that allows you to change the song using a simple gesture such as a twist of their wrist.

Moving onto the objective, both the 1st and 2nd objectives were completed in the initial reading Chapter 3, the literature review. The next objective, the application of gesture control was completed by all the initial applications, prototypes and the finished library in all 3 dimensions. Objective 4 a distributed system, in the end, was not needed. The website maxhunt.github.io was created to allow for this if a need for this has arisen.

Objective 5 was to meet the Pebble’s memory requirement by generating/developing efficient code. As seen from the various testing and requirement evaluation sections this was also met, but not without trouble along the way. If this had not been met, the overall functionality of the project would have been flawed and may have made the Gebble.js unusable.

Meeting number 6 was also crucial as if this was not met, I would be breaking the law. The data protection act, was followed and no infringements took place. All problems highlighted in the Privacy section of the literature review were considered throughout the project to help this.

Finally, objective 7 highlighted the release of the project. Although nothing was released to the Pebble Store, as the applications produced are not applicable to it, Gebble.js and all the applications have been released from my GitHub. Future improvements, outside this projects time scale, may be released and applications using this library certainly will.

7.2 Future development

The intention is to carry on development of Gebble.js as a personal project. Here are the ideas and future development paths which have been thought about, or suggested during a student conference that was attended.

7.2.1 Doing more for the developer

Whilst the current implementation of Gebble.js is very usable by the developer, could be made improve their experience with Gebble.js. However, this would require research into the user base. This is something that could only happen after developers have had experience with the current release version. I would be asking for feedback on the current way they interact with the Gebble.js. Would the developers want more done for them, at the loss of some control over the accelerometer and at the expense of more memory used by Gebble.js.

Currently an event from the accelerometer is passed to the library and acted upon. I am fully aware that some developers would wish to subscribe to an event supplied by the library rather than supply one to it. This would require a loss of control over the accelerometer. The initialisation of the accelerometer and control of the accelerometer would take place in the library. A start and stop function would then control whether the application looking for a detection within the gesture, improving the ease of use of Gebble.js to the more inexperienced developer.

The drawbacks are that a more experienced developer may want to use the accelerometer for something else in their application. This would mean the accelerometer could be initialised twice and this would not only be inefficient but may cause the 2 parts to interfere with each other. This could be solved by producing different versions or packages of the library for different needs, something which is being considered.

Another objective that has been mentioned before, that could also be performed using this above method, is transferability. At this time the library will only work correctly on the Pebble, but the algorithm has been made to work only in JavaScript. A future improvement could be to split the library into 2 parts that make up Gebble.js.

The first part would be the interface with the pebble and possibly the method of subscription above. The second part would be pure JavaScript used to implement gesture control on Gebble.js, but can be used outside of the Pebble environment. Transferability was something that was discussed in the student conference and is requirement 23.

7.2.2 Student Conference

This was attended, to demonstrate my project to people from the business and educational environment. Transferability wasn’t the only future aspect discussed here; other areas such as medical and business were also debated.

The most interesting idea was from a medical perspective; an application to monitor Parkinson's patients. Tremors of the body, commonly in the hand and arm, are one of the symptoms of this disease; this would be monitored by an application using Gebble.js.

It would allow the patients to be monitored by the hospital and doctors without the need for constant check-ups. Not only saving the patient hassle, but also the hospital money with a small start-up cost, of the watch.

This was an unusual suggestion as the gestures would not be acted upon by the user, but instead monitored from by someone on the outside and recorded. The data would only be acted upon if a certain threshold was met. With the ability of multiple gestures being monitored at the same time, this could also monitor the patient for falls and lack of movement over a time period.

All this is made possible by the price of the watch, the ability for communication to an outside service via ajax in JavaScript and of course, Gebble.js.

Many other business, presentation equipment, and gaming applications, augmented reality, were suggested but above seemed the most interesting application.

7.3 Project management

This project has mostly been on track throughout, with the exception of the noise monitor application. This caused a minor delay in the overall project plan meaning a lot more work was completed in January. Therefore an updated gantt chart is supplied (figure 31).

The busy month of January is also reflected by the GitHub contribution calendar, located at https://github.com/MaxHunt. This shows many contributions in the January and early February months, in line with the development process.

7.3.1 Revised Gantt Chart

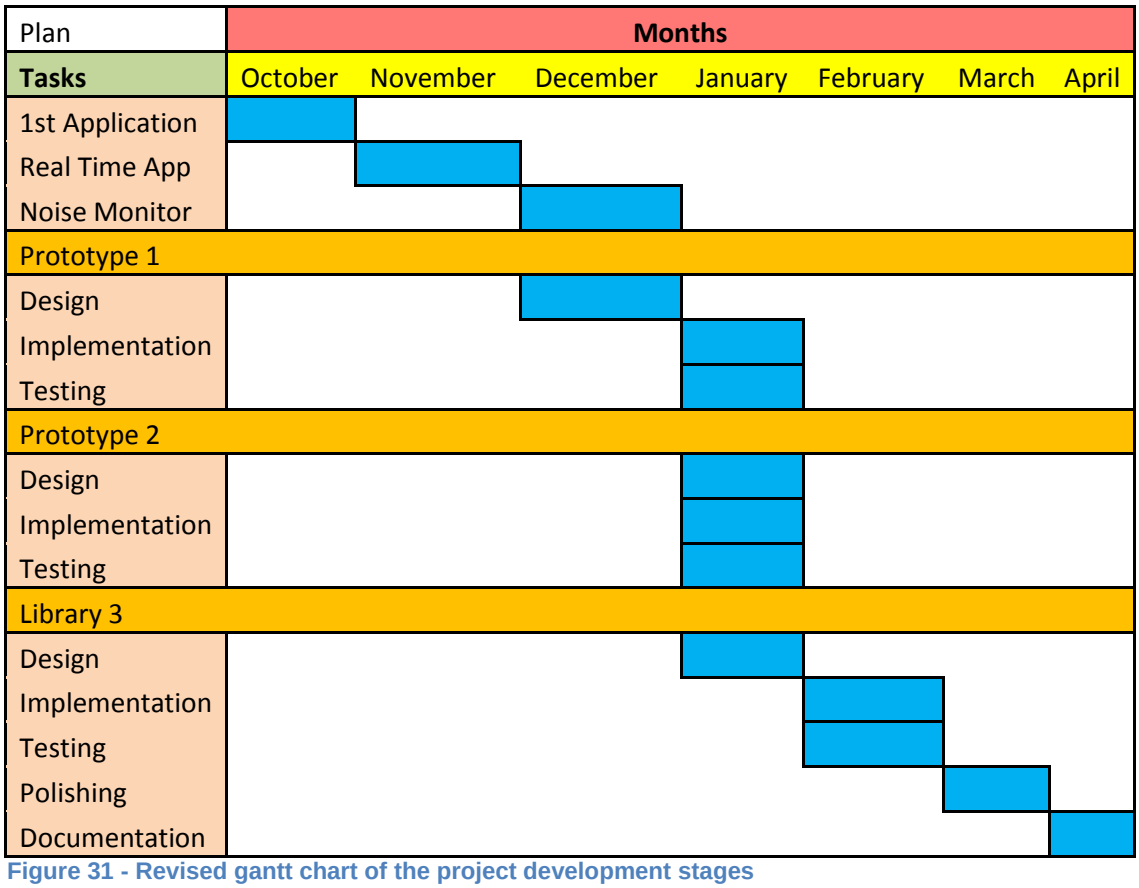


Figure 31 - Revised gantt chart of the project development stages

7.4 Personal Reflection

Thought-out the project I have commented briefly on my thoughts about various sections and parts of the implementation process. This is a general personal conclusion of the whole project.

This may be one of the largest and most significant projects undertaken so far in my career. It was important to find a subject which would interest me to the end. If I did not have interest in what I was doing, how could I expect someone reading about my project to have interest in the subject I was talking about? I choose the right topic and have hopefully conveyed my interest and enthusiasm throughout with the detailed descriptions of the problems faced and analysis of them.

I am also very pleased with the result of this project. Not only have the deliverables met all of the projects requirements, they have also met my own expectations. Gebble.js is very usable and people have demonstrated interest in it in the near future. The student conference also boosted my expectations of the library outside of the Pebble environment.

8 References

1. Brennan, K. (2009). *A guide to the Business analysis body of knowledge (BABOK guide).* Toronto: International Institute of Business Analysis.

2. Davis, A. M. (1993). *Software Requirements: Objects, Functions, & States*. New Jersey, United States of America: P T R Prentice-Hall, Inc.

3. Davis, F. D. (1989). Perceived Usefulness, Perceived Ease of Use, and User Acceptance of Information Technology. *MIS Quarterly, 13*(3), 319-340. http://www.jstor.org/stable/249008

4. De Hert, P., & Papakonstantinou, V. (2012). The proposed data protection Regulation replacing Directive 95/46/EC: A sound system for the protection of individuals. *Computer Law & Security Review, 28*(2) , 130-142. http://dx.doi.org/10.1016/j.clsr.2012.01.011

5. Du Bois, E., & Gerritsen, B. D. M. (2013). Demonstration of software concepts to multiple stakeholders using modular abstract prototyping. *CoDesgin, 9*(3), 137-159. http://dx.doi.org/10.1080/15710882.2013.824485

6. *Forecast market value global wearable technology market 2012-2018*. (2015). Retrieved from the Statista website: http://www.statista.com/statistics/302484/wearable-technology-market-value

7. Glinz, M. (2007) On Non-Functional Requirements. *15th IEEE International Requirements Engineering Conference.1,* 21-26. http://dx.doi.org/10.1109/RE.2007.45

8. Hackenberg, G., McCall, R., & Broll, W. (2011) Lightweight palm and finger tracking for real-time 3D gesture control. *Virtual Reality Conference, 2011 IEE,* 19-26. http://dx.doi.org/10.1109/VR.2011.5759431

9. Hu, P. J., Chau, P. Y. K., Lui Sheng, O. R., & Tam, K. Y. (1999). Examining the Technology Acceptance Model Using Physician Acceptance of Telemedicine Technology. *Journal of Management Information Systems, 16*(2), 91-112. Retrieved from http://www.jstor.org/stable/40398433?origin=JSTOR-pdf

10. Kela, J., Korpipää, P., Mäntyjärvi, J., Kallio, S., Savino, G., Jozzo, L., & Marca, S. (2006). Accelerometer-based gesture control for a design environment. *Journal Personal and Ubiquitous Computing, 10*(5), 285-299. http://dx.doi.org/10.1007/s00779-005-0033-8

11. Kirkham, R., & Greenhalgh, C. (2015) Social Access vs. Privacy in Wearable Computing: A Case Study of Autism. *Pervasive Computing, IEEE, 14*(1), 26-33. http://dx.doi.org/10.1109/MPRV.2015.14

12. Kunze, E. (2015). *Report: Top 10 Smartwatch Manufacturers By Sales (Q4 2014).* Retrieved from https://wtvox.com/2015/03/report-top-10-smartwatch-manufacturers-by-sales-q4-2014

13. Legris, P., Ignham, J., & Collerette, P. (2003). Why do people use information technology? A critical review of the technology acceptance model. *Information and Management, 40*(3), 191-204. http://dx.doi.org/10.1016/S0378-7206(01)00143-4

14. Losada, B., Urretavizcaya, M., & Fernández-Castro, I. (2013). A guide to agile development of interactive software with a "User Objectives" -driven methodology. *Science of Computer Programming, 78* (11), 2268–228. http://dx.doi.org/10.1016/j.scico.2012.07.022

15. McDonald, J. (2015). *Wearable technology and privacy - the top three insights from regulators and policymakers*. Retrieved from http://www.computing.co.uk/ctg/opinion/2395029/wearable-technology-and-privacy-the-top-three-insights-from-regulators-and-policymakers

16. *Methodologies*. (n.d.). Retrieved from the Datamatics Group website: http://datamaticsgroup.com/methodologies/

17. *Noise.* (2015). In *The Free Dictionary.* Retrieved from http://www.thefreedictionary.com/noises

18. O'Flaherty, K. (2014, October). Goggle Glass: a ticking time bomb?. *SC Magazine: For IT Security Professionals,* 22-24. Retrieved from http://www.scmagazineuk.com/google-glass-a-ticking-time-bomb/article/367264/

19. Overview for ARM Cortex-M3 processor. (n.d.) Retrieved from Texas instruments: http://www.ti.com/lsds/ti/arm/arm_cortex_m_microcontrollers/arm_cortex_m3/overview.page

20. Öztürk, K. (2013). Selection of appropriate software development life cycle using fuzzy logic. *Journal of Intelligent & Fuzzy Systems, 25*(3), 797-810. http://dx.doi.org/10.3233/IFS-120686

21. Park, L. T., Hwang, H. S., & Moon, I. Y. (2014). Study of Wearable Smart Band for a User Motion Recognition System. *International Journal of Smart Home, 8*(5), 33-44. http://dx.doi.org/10.14257/ijsh.2014.8.5.04

22. Robertson, J., & Robertson, S. (1994). *Complete Systems Analysis.* 353 West 12th Street New York: Dorset House Publishing Co., Inc.

23. Silva, È. S., & Rodrigues, M. A. F. (2014) Design and Evaluation of a Gesture Controlled System for Interactive Manipulation of Medical Images and 3D Models. *Journal on Interactive Systems, 5*(3), 53-65. Retrieved from http://www.seer.ufrgs.br/jis/article/viewFile/50376/32555

24. Tran, K. T. M., & Oh, S. H. (2014). A Hand Gesture Recognition Library for a 3D Viewer Supported by Kinects Depth Sensor. *International Journal of Multimedia & Ubiquitous Engineering, 9*(4), 287-308. http://dx.doi.org/10.14257/ijmue.2014.9.4.31

25. Tyrer, A. (2015). Can the UK cyber-security industry lead the world?. *Computer Fraud and Security, 2015*(2), 5-7. http://dx.doi.org/10.1016/S1361-3723(15)30006-3

26. Wang, X., Zhang, X., & Dai, G. (2007) Tracking of Deformable Human Hand in Real Time as Continuous Input for Gesture-based Interaction. *IUI' 07,* 235-242. http://dx.doi.org/10.1145/1216295.1216338

27. Wu, J., Pan, G., Zhang, D., Qi, G., & Li, S. (2009) Gesture Recognition with a 3-D Accelerometer. *Ubiquitous Intelligence and Computing,* 25-38. http://dx.doi.org/10.1007/978-3-642-02830-4_4

28. Yang, H. D., Lee, J., Park, C., & Lee, K. (2014). The Adoption of Mobile Self-Service Technologies: Effects of Availability in Alternative Media and Trust on the Relative Importance of Perceived Usefulness and Ease of Use. *International Journal of Smart Home, 8*(4), 165-178. http://dx.doi.org/10.14257/ijsh.2014.8.4.15

29. Zhang, X., Lv, S., Xu, M., & Mu, W. (2010). Applying evolutionary prototyping model for eliciting system requirement of meat traceability at agribusiness level. *Food Control, 21*(11), 1556-1562. http://dx.doi.org/10.1016/j.foodcont.2010.03.020

Appendix A – PID

School of Computing final year project

Max Hunt

PJE40

Project Initiation Document

The Application of Gesture Control and Quick Response Techniques on the Smart Watch Platform, using JavaScript

1. Basic details

Student name:	Max Hunt
Draft project title:	The Application of Gesture Control and Quick Response Techniques on the Smart Watch Platform, Using JavaScript
Course:	Bsc Computer Science
Client organisation:	N/A
Client contact name:	N/A
Project supervisor:	Jacek Kopecky

2. Outline of the project environment and problem to be solved

Wearable technology is becoming ever more present on the market, if sometimes expensive, and is now available to be purchased in the public domain. However, with this new technology, I see a huge drawback and potential for improvement. Ease of use. People are forevermore trying to automate and improve the ease of their life, for instance, voice activated GPS and escalators. This also proved by the emergence of this technological sector. All these allow us to get to places or interact with services requiring minimal effort and concentration.

Although wearable technology is improving in that aspect, all still use touch for interaction which can be frustrating of slow in comparison to another form, gesture control. For example, if you wish to say “Hello” to someone from a long distance in which they are inaudible to you, a gesture would be used. This could be something as simple as a wave. I would like to apply this natural and quick process of thinking to wearable technology.

The focus of my project is to improve the interaction between the user and their smart watch, to the point where the gesture control becomes easier and quicker than using the buttons or touching the screen.

I would like the end result of my project to influence future development on the smart watch platform; specifically ease of use and, if the research is successful, gesture control.

3. Project aim and objectives

The overall aim of the project is to improve the ease of use on smart watch platforms, using JavaScript as the developing language.

In order to achieve this I will be completing these objectives:

- Investigate **Gesture Control** with users to find unique and easy gestures to apply.
- Research and scrutinize different **Gesture Recognition** techniques that have already been implemented and identify if any current development is adaptable or transferable to JavaScript.
- Explore and apply **Gesture Control** to the Pebble using the 3D accelerometer.
- Consider the creation of a website to create a **Distributed System** across the platform to store data and/or application features.
- Meet the Pebble's memory requirement by generating **Efficient Code**.
- Research and, if relevant, include basic **Text Analysis** one of the applications to help with ease of use.
- Make sure all according and data collected it in accordance with the **Data Protection Act**.
- Release the application to the market

4. Project deliverables

The deliverables I hope to create during this project are:

- A Library for Gesture Control
- A number of native applications for the Pebble.
-

Additionally, depending on the outcome of initial research, I will produce the system artefacts:

- A website interfacing with the applications; used to store data and for additional computing power if needed
- A Library for Text Analysis

Finally, documentation will be produced. A report will be created outlining, discussing and evaluating the project. Documentation on the code/Library will also be made available.

5. Project constraints

With every project there are constraints and this project is no exception. Undoubtedly, I have the generic constraints of time; the project has only 9 months, workload; I have other units and coursework that I need to complete.

However, I have other constraints that are unique and somewhat uncommon in the now very much evolved, industry of technology; the most pronounced being memory availability, but also industry evolution and the Pebble SDK platform.

The Pebble, being a small gadget, has very limited space for memory storage. As a consequence, it not only has a much smaller available space for volatile memory but also non-volatile. The pebble can only have 8 apps stored on it at a time, each with a maximum memory allowance. A couple of solutions have already been thought of, such as a native app or storage in a database on a website.

Additionally, with the industry of technology and its constant evolution, the wearable device sector is growing very quickly. This could potentially mean that the platform I have opted for could be a legacy device by the time of this projects completion.

The final constraint, applicable to just this project, is that the Pebble SDK is only available on MAC and Linux machines, in offline mode. This means that if I wish to develop offline, I must have one of these systems.

As with all technology and the potential for data collection, the Data Protection Act must not be ignored. If I do venture into the text analysis sector, I may come across constraints applied by this act.

6. Project approach

I am going to approach my project with an open mind. By this I mean that I am clear by what I want to do and achieve but am very open for the project to go into another direction should a greater problem recognised.

I will achieve my project aim by applying gesture control and quick response techniques to a number of applications. These will start small scale and become more intricate as I develop my gesture control library. The current ideas are: A small, quick response system with either a “Yes” or “No” response; music control system; gesture tracking which is then mapped onto the screen.

I shall be using the **Agile** method to approach my project with a **Sprint** method in place to achieve my goals and prototypes of applications. A sprint will be based on development and allows me to create concrete milestones and easily track progress. These will be over a 2 week period.

My skills as a developer are honed enough to progress in this project, but I have limited knowledge with JavaScript. This will be addressed by the taking of the Graphics Vision Unit, but I will need to be learning how to use this language to its full capability over the first couple of months of my project.

I will admit one of my weaknesses is efficient coding. All though the end result will be the same; my code may not be as good as it could be. This will also need to be worked on as

personal goal. I can correct this by reading into the subject, as well as trial and error during development.

7. Facilities and resources

I do not have access to a Mac. I will therefore be using one at the university; this allows me to develop in offline mode. The mobile lab has many Macs but also has Apple development kits, should the need arise to use them. The mobile lab is sometimes in use but there will be times when I can go in.

If this need does arrive to use the development kits, due to the Pebble being available on multiple platforms, I will also be developing on the Android software. This brings the necessity for a phone using this software. Fortunately, I have access to one of these.

Obviously, naming the Pebble as my primary platform, I will be using the Pebble smart watch. I will use the “Cloud Pebble” service, which is linked to GitHub, for development.

8. Log of risks

Risk Description and Type	Risk Impact	Risk Probability Low - Medium - High	Mitigation / control	First indicator
Time – Running Out of time for Hand-in	No completion of artefact	Low	Amend/ Monitor Methodology and milestones	Milestones missed
Pebble Breaks	Nothing to test on	Medium	Be careful with it.	Non-Responsive testing
Loss Of Work	Nothing to Hand-in	Low	Save in multiple locations regularly. Dropbox.	No access to work.
Illness	Have to Defer year	Medium	Keep good health	Ill Health
Evolution of market	Pebble becomes legacy	Medium	Keep Updating Software and keep with current trends in Application development	No new updates
Inexperience at JavaScript leads to inefficient coding	Applications don’t run smoothly	High	Constant testing of code for memory leaks	Memory Leaks

9. Starting point for research

I so far have some journal articles:

Affordances of Touch in Multi-Sensory Embodied Interface Design, *Simone Gumtau* (2011)

Activity Detection Using Frequency Analysis and Off-the-shelf Devices, *Sebastian Bersch* (2013)

Real-Time Human Ambulation, Activity, and Physiological Monitoring: Taxonomy of Issues, Techniques, Applications, Challenges and Limitations, *Ifeyinwa Achumba* 2013

A Grounded Theory of Emergent Benefit in Pervasive Game Experiences, *Neil Dansey* (2013)

I have also done some research into application and libraries on the pebble that involve accelerometer use. There is an application already in use that could allow me to create a chart, over all three axis, for a movement.

10. Breakdown of tasks

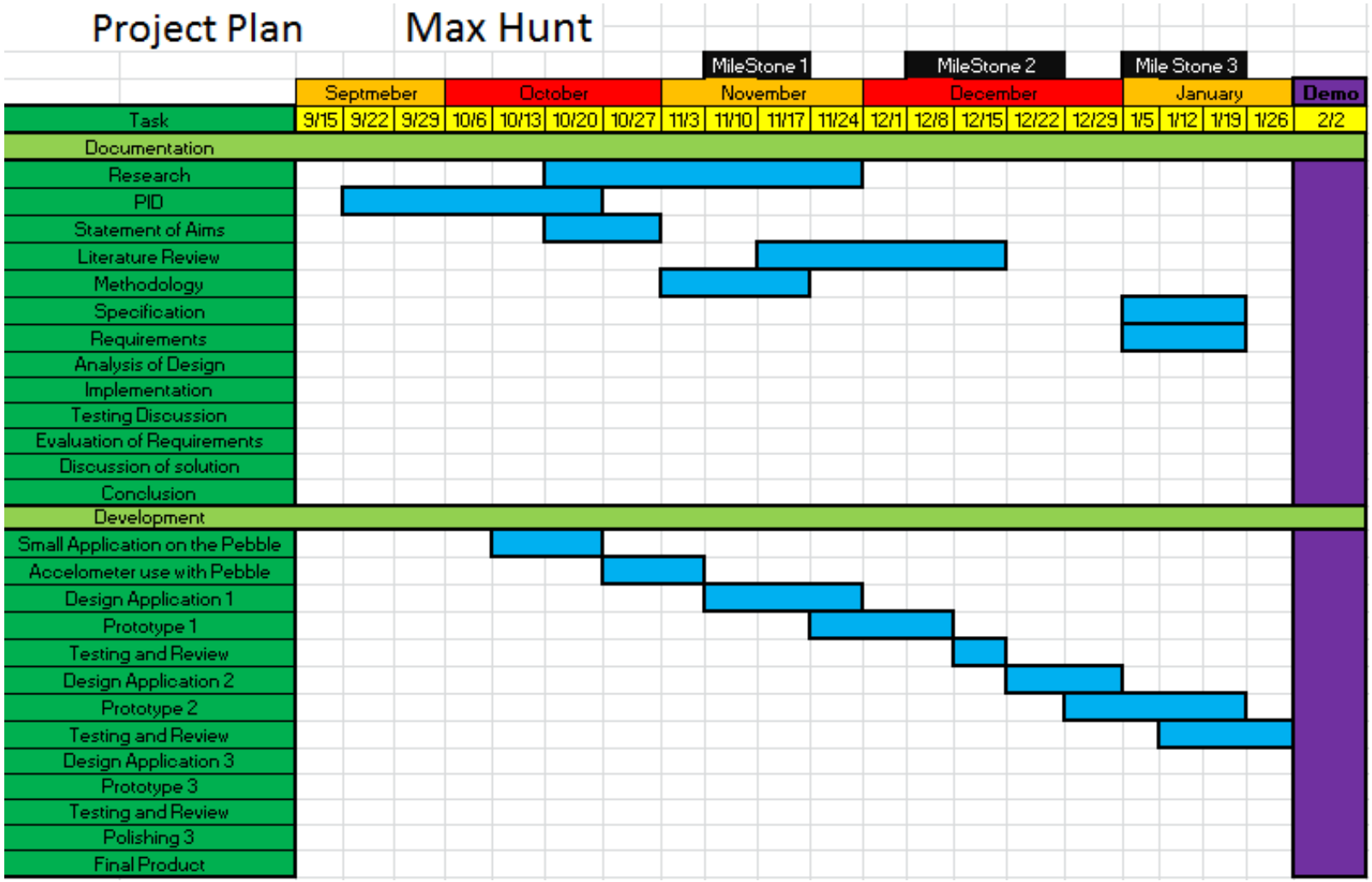
As stated I propose to work in an agile method. This will allow me to adapt my project as it goes on and prototype my work constantly. However, In order to do this, I will be completing a lot of documentation about project approach e.g. Specifications and requirements. I will also be working on the literature review over a long period to allow for many drafts. Here is list of tasks I wish to complete (Un ordered):

- Documentation
    - Research
    - PID
    - Statement of Aims and Objectives
    - Literature Review
    - Methodology
    - Specification and Requirements

    - Analysis of Design
    - Implementation
    - Testing Discussion
    - Evaluation of Requirements
    - Discussion of Solutions
    - Conclusion
- Development
    - Small application for the Pebble
    - Accelerometer use in the Pebble
    - Design for Prototype 1
    - Development 1
    - Testing and Review
    - Design for Prototype 2
    - Development 2
    - Testing and review
    - Design for Prototype 3
    - Development 3
    - Testing and Review
    - Polishing 3
    - Final demo of 3

11. Project plan

This plan will be monitored and maintained. This is subject to change. The plan has so far been made to up to first demo.



12. Legal, ethical, professional, social issues

As mentioned in the constraints section the Data Protection Act could have an impact on this project as well as the new cookie laws (UK Only).

If I do gather data of people gestures, is this a violation of the Data Protection Act. The problem is that this is a rather grey area. Although not directly linked to them in any personal way, it is still data on their movements that I am collecting. This will require more research.

However, it is abundantly clear that if I have an element of text analysis, I will be reading people incoming messages. This data is will not be stored by myself of the application. But the user will have to be made aware that this is happening.

Ethical Checklist attached.

Signatures

	Signature:	Date:
Student		
Client		
Project supervisor		

Appendix B – Ethics Certificate

Certificate of Ethics Review

Project Title:	Improving the Ease of Use on Smart Watches, with Gesture Control
User ID:	609556
Name:	Max Edmund Hunt
Application Date:	16/04/2015 13:41:25

You must download your referral certificate, print a copy and keep it as a record of this review.

The FEC representative for the School of Computing is Carl Adams

It is your responsibility to follow the University Code of Practice on Ethical Standards and any Department/School or professional guidelines in the conduct of your study including relevant guidelines regarding health and safety of researchers including the following:

- University Policy
- Safety on Geological Fieldwork

It is also your responsibility to follow University guidance on Data Protection Policy:

- General guidance for all data protection issues
- University Data Protection Policy

SchoolOrDepartment: SOC
PrimaryRole: UndergraduateStudent
SupervisorName: Dr Jacek Kopecky
HumanParticipants: No
PhysicalEcologicalDamage: No
HistoricalOrCulturalDamage: No
HarmToAnimal: No
HarmfulToThirdParties: No
OutputsPotentiallyAdaptedAndMisused: No
Confirmation-ConsideredDataUse: Confirmed
Confirmation-ConsideredImpactAndMitigationOfPontentialMisuse: Confirmed
Confirmation-ActingEthicallyAndHonestly: Confirmed

Certificate Code: E573-FC8C-3534-E8F5-0174-E2CC-42F1-B0FC Page 1

Supervisor Review

As supervisor, I will ensure that this work will be conducted in an ethical manner in line with the University Ethics Policy.

Supervisor signature:

Date:

Appendix C – Test Plans

1. Non-Functional Requirement 14 Test Plan

False Positives are tested by completing gestures and movements that should not be identified as the gesture that is being searched for. E.g. circular movements should not be identified as a shakes. A range of movement will be tested to full fill this requirement. Overall Result is “Pass” at 93.3%.

Test No.	Testing	Expected Result	Actual Result	Conclusion
1	Circular movement	No Detection	No Detection	Pass
2	Circular movement	No Detection	No Detection	Pass
3	Circular movement	No Detection	No Detection	Pass
4	Circular movement	No Detection	No Detection	Pass
5	Circular movement	No Detection	No Detection	Pass
6	Y Directional Movement	No Detection	No Detection	Pass
7	Y Directional Movement	No Detection	No Detection	Pass
8	Y Directional Movement	No Detection	No Detection	Pass
9	Y Directional Movement	No Detection	No Detection	Pass
10	Y Directional Movement	No Detection	No Detection	Pass
11	X Directional Movement	No Detection	No Detection	Pass
12	X Directional Movement	No Detection	No Detection	Pass
13	X Directional Movement	No Detection	No Detection	Pass
14	X Directional Movement	No Detection	No Detection	Pass
15	X Directional Movement	No Detection	No Detection	Pass
16	Z Directional Movement	No Detection	No Detection	Pass
17	Z Directional Movement	No Detection	No Detection	Pass
18	Z Directional Movement	No Detection	No Detection	Pass
19	Z Directional Movement	No Detection	No Detection	Pass

20	Z Directional Movement	No Detection	No Detection	Pass
21	Random Movements Light	No Detection	No Detection	Pass
22	Random Movements Light	No Detection	No Detection	Pass
23	Random Movements Light	No Detection	No Detection	Pass
24	Random Movements Light	No Detection	No Detection	Pass
25	Random Movements Light	No Detection	No Detection	Pass
26	Random Movements Hard	No Detection	Detection	Fail
27	Random Movements Hard	No Detection	No Detection	Pass
28	Random Movements Hard	No Detection	No Detection	Pass
29	Random Movements Hard	No Detection	Detection	Fail
30	Random Movements Hard	No Detection	No Detection	Pass

2. Non-Functional Requirement 17 Test Plan

This test was completed with a timer. It was timed from the end of a completed gesture to the output of a detection “true variable”. Shake are being used her as they are the easiest gesture to complete. Over all result was a “Pass” at an average of 0.733 seconds.

Test No.	Testing	Expected Result	Actual Result	Conclusion
1	Shake	Detection < 2 seconds	0.28 seconds	Pass
2	Shake	Detection < 2 seconds	0.56 seconds	Pass
3	Shake	Detection < 2 seconds	1.32 seconds	Pass
4	Shake	Detection < 2 seconds	1.03 seconds	Pass
5	Shake	Detection < 2 seconds	0.78 seconds	Pass
6	Shake	Detection < 2 seconds	0.81 seconds	Pass
7	Shake	Detection < 2 seconds	0.41 seconds	Pass
8	Shake	Detection < 2 seconds	1.05 seconds	Pass
9	Shake	Detection < 2 seconds	0.96 seconds	Pass
10	Shake	Detection < 2 seconds	0.13 seconds	Pass

Appendix D – Documentation for Gebble.js

Gebble.js is a library which helps developers implement gesture control into JavaScript applications, using the Pebble smart watch and pebble.js.

1. Setting up Gebble.js

First of all you must add the Gebble.js library to your files in the application. Gebble.js can be found here, https://github.com/MaxHunt/Gebble. The “read me” is not required for Gebble.js to work.

Secondly, you must require Gebble.js in the files you wish to use it.

```
//include Gebble Library

var Gebble = require('Gebble');
```

After this, you can use the Gebble.js functions. However, you must initiate the accelerometer in your application. The details for this can be seen in pebble’s documentation: http://developer.getpebble.com/docs/pebblejs/ . The optimum setting for gesture detection is 25 samples at 10Hz. This is also the setting the default gestures have been defined and tested with.

2. Using the functions

To start use the init() function, this allows you to set options for the Gebble.js. Here is an example of how this works. They are passed as an object to the init function.

```
//gestures

var shake =
[[{x:3000,y:3000,z:3000,name:"shake"},{x:100,y:100,z:100}],[{x:3000,y:3000,z:300
0},{x:100,y:100,z:100}],[{x:3000,y:3000,z:3000},{x:100,y:100,z:100

var shakeX =[[{x:4000,y:4000,z:4000,name:"shakeX"},{x:1100,y:0,z:0}]];

var shakeY =[[{x:4000,y:4000,z:4000,name:"shakeY"},{x:0,y:1100,z:0}]];

var shakeZ =[[{x:4000,y:4000,z:4000,name:"shakeZ"},{x:0,y:0,z:1100}]];

var gestures = [shake,shakeX,shakeY,shakeZ];

//start APP

var opt = {debug:true, delay:3000, gestures:gestures};

Gebble.init(opt);
```

“Debug” allows the output of messages in the console telling you when detection has been made, among other useful information. Can be true or false

“Delay” is the amount of time another gesture can be detected from a previous positive detection, the cool down time if you will. The 3000 value above represents 3 seconds. An integer.

“Gestures” is the array of gestures you wish to detect. In the example, there are 4 different gestures. The order in the array denotes the priority of the gesture. If first, this gesture compared first and so on.

The gesture format is that of an array. Within the array are number of frames, also each an array. These frames contain 2 objects, each with 3 variables, excluding the first object in the array, which contains a name field.

The 2 objects contain upper and lower tolerances for comparison to the measurements coming into the algorithm. The maximum is 4000 and minimum 0 as they differences are calculated absolutely. The tolerance is compared to the difference of a frame on the measurement side. For example if x1-x2 > g1 this is evaluated to true.

All these options have default values. “Gesture” defaults to a shake, X Movement, Y Movement and Z movement, “Delay” to 3000 and Debug to “true” if no input is found.

In order to use Gebble.detect(), you must supply it with the event given to you by the accelerometer. An example below.

```
var detection = Gebble.detect(e);
```

The response will be either true or false, depending on whether a gesture is detected in that event. You will also receive an integer relating to the position of the gesture in the array supplied of the default array, as explained above. If the detection was false, this number will be -1.

3. Example Project

An example project using Gebble.js is here https://github.com/MaxHunt/StarWarsGen.

This is also the project the above snippets of code were taken from. This project was adapted from https://github.com/dcgauld/starwars.

Any other questions, please feel free to tweet me @MaxHnt, or contact me on Github.

Appendix E – Other Applications

This appendix describes the design and implementation of the first 2 apps I created for this project.

1. The First Application

After the decisions made at the start of chapter 7, the first application was started. The design decisions were mainly avoided here as, at the start of the project, I was highly inexperienced with JavaScript and did not know much about Pebble's API either. This was an exploratory application.

However, the overall aim of this application was to produce something that could output a readout of the accelerometer, on a button click. This would pave the way for the next application, a real-time readout. 7.1.5 explains this application.

This Snapshot application, "AccelSnapShot" as I called in in the final Development stage, was a challenge but surprisingly from the Graphical User Interface side. It was fairly simple to get readout of the data into a log to the console, but placing it on anything other than the start-up screen of the application was a challenge.

Figure 32 - Result Screen of the "AccelSnapShot" application

During the limited designing stage of this app, a decision had been made to have the start-up screen display instructions on how the app worked and then a second screen would appear (figure 32), after a button press. This screen would display the snapshot information from the accelerometer at the time of the button press. When trying to implement this I found that Pebble uses a rather, in my opinion and admittedly limited experience, unconventional method of GUI handling.

The displayed windows can stack on top of each other, after changing to a screen further into the application. But, if a user uses the back button, to come out of a lower level of the application, this does not then close the screen; this only "hides" the window. This leads to multiple instances of said window being left open, unless the button is overwritten with a close function that implements the same command twice, the command that hides window. This is the method that is listed in the Pebble documentation.

Whilst this was easily fixed by the aforementioned technique, later on a similar bug with the Noise Monitor Application arose, which wasn't so easy to fix. When unhandled, this problem causes a memory leak which is never a welcome problem, but especially with the Pebbles limited memory availability.

Despite this problem, I have met many of my requirements with this application, the number of these completed requirements are as follows: (Functional) 11,(Non-Functional) 1,4,5,6,7,8,9,12,16,19 and 21. The seventh Non-functional requirement asks to adhere to memory usage percentage of 7/8 or 87.5%. This apps usage was calculated to 20.5%, well below the required level. Number 12 requires an icon to pass, this can be seen in Figure 11, next to the applications name.

All the additional un-met requirements of this application are not applicable to it and therefore are of no concern, except for number 13. This requires the application to be released via the Pebble store. Although this could have been done, a release version published on GitHub was seems satisfactory for the initial application which was meant to test myself as a developer rather than be released for public use.

2. The Next Step – A Real Time Monitor

After the learning curve from the “AccelSnapShot”, the next stage obvious stage in development, towards my first prototype, was the ability to monitor and read the accelerometer data in a real time capacity. A separate application was conceived to try various techniques before deciding on the most efficient way to handle the event driven system of the pebble.js API.

Figure 33 - Screen Shot of Real Time Application

The first design executed was that similar to the previous, but using a loop to continuously update the screen with the firing of a simulated event. Whilst working in design and theory, upon the first tests, the length the application could run was under a minute before crashing. I found that this was due to the a-synchronous nature of JavaScript. A new loop was being created on each event, without the previous one ever timing out, eventually leading to the memory on the pebble being used up, crashing the Pebble App and watch crashing.

This caused the change in direction to the application format, which continued on throughout my project. After initially looking through the Pebble documentation for a fix with the GUI, an event was found that fired every time the accelerometer handler filled the number of samples it was allocated (default 5). The accelerometer objects could also be subscribed too and unsubscribe from. This gave control that was needed to produce real-time input, which could be stopped at any point and display it as well, without the need for a loop. Every time the event fires it takes the data into a function and displays this once. This function is then discarded and the application waits for the next event.

At the same level as the accelerometer event, it is also checking for a back button press. Whilst, the back function is already automatic, the application needed to unsubscribe from the accelerometer object to stop the events firing whilst not on the output screen, saving precious memory. Whilst not so important in this application, it was good practice as well as future proofing the application.

Decided during this application was the rate of sampling and the amount of samples. At a rate of 10 Hz, to 25 samples, the application could keep up with incoming data. If this was decreased a visible gap in the measurement could be seen. If increased the events would start to pile up as they could not be processed quick enough. These variables are in line with the predicted requirement.

This application also met a number of the requirements that where outlined. Functional requirements completed: 7. Non-Functional Requirements completed: 1, 4, 5, 6, 7, 8, 9, 12, 13, 16, 19 and 21. As the previous application, the seventh Non-functional requirement asks to adhere to memory usage percentage of 7/8 or 87.5%. This apps usage was calculated to 21%, below the required level. Number 12 requires an icon to pass this can be seen in Figure 33, next to the applications name.

Furthermore, all the additional un-met requirements of this application are not applicable to it and therefore are of no concern. However, as with the application before, it was decided not to release this via the pebble store, as it was more an exploratory application for me rather than a consumer application. However, it has been professionally finished and has been released on my twitter and website with a link to the release version on my GitHub.